Visual Co-occurence Learning using Denoising Autoencoders

Victor Deleau

A thesis
in
The Department
of
Computer Science and Software Engineering

Presented in Partial Fulfillment of the Requirements
For the Degree of Master of Computer Science
Concordia University
Montréal, Québec, Canada

August, 2020

CONCORDIA UNIVERSITY
School of Graduate Studies

This is to certify that the thesis prepared

By: Victor Deleau

Entitled: Visual Co-occurence Learning using Denoising Autoencoders

and submitted in partial fulfillment of the requirements for the degree of

Master of Science (Computer Science)

complies with the regulations of the University and meets the accepted standards with respect to originality and quality.

Signed by the final examining committee:

_____ Chair

Dr. D. Pankratov

_____ Examiner

Dr. D. Pankratov

_____ Examiner

Dr. A. Delong

_____ Thesis Supervisor(s)

Dr. J.Y. Yu

_____

Approved by _____

Dr. L. Kosseim Chair of Department or Graduate Program Director

_____

Dr. M. Debbabi Dean, Faculty of Engineering and Computer Science

Date September 9, 2020

Abstract

Visual Co-occurence Learning using Denoising Autoencoders

Victor Deleau

Modern recommendation systems are leveraging the recent advances in deep neural networks to provide better recommendations. In addition to making accurate recommendations to users, we are interested in the recommendation of items that are complementary to a set of other items. More specifically, given a user query containing items from different categories, we seek to recommend one or more items from our inventory based on latent representations of their visual appearance. For this purpose, a denoising autoencoder (DAE) is used. The capacity of DAEs to remove the noise from corrupted inputs by predicting their corresponding uncorrupted counterparts is investigated. Used with the right corruption process, we show that they can be used as regular prediction models. Furthermore, we measure experimentally two of their specificities. The first is their capacity to predict any potentially missing variable from their inputs. The second is their ability to predict multiple missing variables at the same time given a limited amount of information at their disposal. Finally, we experiment with the use of DAEs to recommend fashion items that are jointly fashionable with a user query. Latent representations of items contained in the user query are being fed into a DAE to predict the latent representation of the ideal item to recommend. This ideal item is then matched to a real item from our inventory that we end up recommending to the user.

Acknowledgements

I would like to thank my supervisor Jia Yuan Yu for allowing me to conduct my Master's degree at Concordia. I learned greatly under his supervision, and am now a much better researcher than I used to be. I am also grateful to Volker Haarslev for giving me a chance to prove my worth before transferring to the Computer Science department, and more generally to the Concordia staff for their professionalism.

Thank you to Laurent Charlin for validating and encouraging the approach taken in this work. Many students in the lab and beyond helped me improve and finalize this thesis. Special thanks to Syed Eqbal, Denis Ergashbaev, and Tyler Manning.

I thank my uncles Loïc and Dominique for the model of exemplarity that they represented to me. I also thank my aunts, my grandparents, and my mother Sylvie. They always provided me with love and help. My family in Quebec has also been particularly caring since my arrival, and I deeply thank them for that.

Finally, I could not call Montréal home without my friends Valentin Fleury, Mèét Patel, my cousin Sebastien, as well as the many other amazing people that I had a chance to meet. Special thanks to William Lim for having relentlessly watered my plants during the Covid19 pandemic.

Contents

List of Figures

1. Introduction

The massive economic growth of the 20th century brought people more freedom and wealth. We can now buy many goods for increasingly competitive prices, representing an increase in people's buying power. But this has also led to a surge in the number of choices people have to make on a day-to-day basis. Customers are facing the problem of finding what they need in physical and online stores, and potential sales are sometimes not realized because of the customer's inability to find what they need. From the retailer's perspective, competition is getting fiercer with an increasingly polarized market, which calls for more value proposition and differentiation. Retailers are facing the challenge of satisfying increasingly demanding customers who want to find exactly what they have in mind and fast.

Those challenges are all but present in the fashion industry. With a compound annual revenue growth rate of 7.5% per year from 2014 to 2018 [13], the fashion industry is on the rise. Everybody buys clothing, and nearly all of us are concerned about the way we look and what society thinks about it. Being able to dress well, for the right situation, is perceived as a quality and makes one more attractive. Few of us master this skill, yet most would like to look better, which highlights the great innovative potential that remains untapped. The fashion industry is getting increasingly competitive, with a winner-take-all type of market that is pushing small fashion retailers out of the race. According to the 2019 McKinsey technical report on the fashion industry [53], 97% of the profits in the fashion industry are now earned by only 20 companies, and most of them have doubled their profit during the last decade. So what are exactly the strategies put into practice by fashion retailers to adapt to this new reality?

One of the ways fashion retailers have been coping with those difficulties is by developing recommendation systems. By collecting data on their customer's behaviors, like historical purchases or implicit interactions, recommendation systems can propose a set of items that a customer would be more likely to appreciate and buy. First introduced in the 90's, recommendation systems give an edge to the companies that are using them. They substantially increase the turnover of companies. For instance, 35% of what is bought on Amazon and 75% of what is watched on Netflix now come from recommendations made by such algorithms [48]. Therefore, recommendation systems are now central to the success of not only fashion retailers but of all online retailers.

Originally based on traditional machine learning techniques, the field of recommendation systems is now benefiting from the latest development in artificial intelligence with the advent of deep neural networks. Autoencoders, a specific architecture of neural networks, highlight one of the major strengths of deep neural networks: their capacity to learn dense and abstract representations of high-dimensional and complex data. One of the domains where neural networks excel is in image-processing. Convolutional neural networks, either applied to image generation or classification, now bring an astonishing prediction accuracy. In the fashion industry, items benefit from rich annotations and people's preferences for them are almost entirely based on their visual appearance. Discrete attributes are in their vast majority just a mere summary of the visual aspect of fashion items. By using convolutional deep neural networks, it is now possible to fully leverage

the visual appearance of fashion items to build better recommendation systems.

Beyond the simple recommendation of items to customers, it is also of interest to learn and recommend what goes well with what in situations where items are observed, bought, or worn together. In this work, we are interested in the problem of recommending complementary fashion items. As this notion of fashionability is not dependent on the user's tastes but rather based on a more general and shared idea, this problem calls for an approach based on the content and appearance of items. Therefore, we leverage the visual aspect of fashion items to extract useful dense representations. We use a particular neural network architecture, a denoising autoencoder (DAE), and experimentally measure two of the characteristics that set it apart from other models, namely their robustness and versatility. We then experiment with the recommendation of complementary items on the Polyvore Outfit dataset and define our own ranking metric to assess the performance of our model.

In this work, we learn a notion of joint fashionability from a set of selected fashion outfits and assume that those fashion outfits are fashionable. This notion of fashionability is strongly related to the concept of style. But what exactly is a style?

1.1 What is a style?

It is well known that some very simple tasks for us humans can be particularly difficult or even impossible for machines to grasp, and conversely hard tasks for us humans can be easily solved by computers. In computer science, this is also known as the paradox of Moravec [1]. A good example of this is the concept of style. Let's consider the style of paintings. At first sight, it is easy for us to distinguish between the style of two different pictorial artists but, until the recent advances in deep neural networks, computers had to combine various visual features extracted using classical techniques like SIFT [51] and SURF [5], as well as clustering algorithms to achieve modest classification results. Indeed, those methods were not able to extract the abstract content that images contain. Deep neural networks, especially applied to vision and language processing tasks, are now able to close this gap.

What we define by style is often highly dependent on the context, but we can still highlight some facts about this concept. First observe that one instance of a fashion style, an outfit, is made of several pieces of clothing that are assembled in a particular way. Each apparel of a given type also has a particular style, which helps us to compare it against other apparel of the same type. Therefore, styles discriminate.

Named styles, like chic or streetwear, are quick and easy to grasp references to something that can be easily perceived and understood by others, but whose attributes are too complex or too large to be described one by one: it appeals to some kind of common understanding of the world. Giving names to some fashion styles facilitates communication. In that sense, a style is also a compressed summary that is easier to communicate and manipulate. New named fashion styles are naturally created to classify and compare different outfits whenever the current set of commonly agreed upon named fashion styles doesn't allow for a good enough characterization of what we want to express.

Neural networks, and more specifically autoencoders, are well suited for the task of understanding and representing styles. Due to their ability to extract dense lower-dimensional representations of their inputs while retaining and organizing most of the information that is characterizing them, it naturally draws similarity with the way we

process and communicate those concepts. This is why, in this work, we use deep neural networks to extract useful representations of the visual appearance of fashion items.

Now that we have defined the concept of a style, we argue about the underlying need that the fashion industry has for a unified way to recommend complementary fashion items. From there, we draw our motivations for this work.

1.2 Motivation for our work

In the last decade, researchers have been considering the problem of recommending complementary items through different angles. However, we believe there is still a significant area of improvement in this area of research.

Many of the methods that were previously proposed were considering pairs of items [29] [68] [41]. Some of them are also unidirectional [41], meaning that they consider an item with a given category as input and output an item from another target category. This implies that a new model must be trained for each pair of potential categories of items and each direction. Given that we might want to recommend items from a large number of different categories, this calls for a more flexible and integrated approach.

Some of the proposed methods are based on collaborative-filtering, a specific type of recommendation system that recommends items to users that similar users to them previously bought. An important issue with collaborative-filtering algorithms, and which they have by design, is their inability to recommend items when no information is known about the user. This is the so-called cold-start problem. Using the purchased history of users to learn the notion of joint fashionability that we are interested in is also problematic as unfortunately, not all users are style aware. Furthermore, co-purchased items do not necessarily imply the joint fashionability of those same items [42]. It is more sound to use a set of outfits that is selected by knowledgeable people and to learn the joint fashionability from the attributes and visual appearance of the items they contain. This would amount to a content-based approach, where customers are being recommended items whose attributes are similar to the one they previously interacted with. Note that there exist a lot of different fashion styles and that a given item could be part of two different outfits whose style is different. Therefore, such a recommendation system could still benefit from the user's input regarding which of those styles the user prefers.

In a content-based recommendation setup, users are being recommended items whose attributes are similar to the one of items they previously bought. For this purpose, discrete attributes like color, category, and textual descriptions are often used. But is it the best way to characterize an item? While it depends on the actual context and type of items we are dealing with, in fashion, we believe that users' behaviors are largely driven by the visual appearance of items. Furthermore, most of those discrete attributes are manually annotated and can be extracted in their vast majority from the visual appearance of items. To a lesser extent, this is also true for home furniture. Directly accessing the source of those discrete attributes could provide richer and more accurate representations of items, which would in turn lead to better prediction accuracy when fed to a prediction model.

Finally, in recent years we have observed new recommendation system models of increasing complexity. This added complexity does not always provide an increase in prediction accuracy [14]. In a real-life setting, several other aspects of a recommendation system might come into play. This includes ease of implementation and practicality of the

model that will be used in production: two things which might lead a decision-maker to not always select the best performing model that is known of. In this work, we recognize the importance of simple but effective approaches to solving problems.

To the best of our knowledge, if some of the previously proposed methods have been able to tackle few of the listed issues, none were able to solve all of them. Building on this knowledge, we now provide our goals for this work.

1.3 Goals

We seek a pure content-based approach that uses the visual appearance of items. As we focus on the recommendation of fashionable items, we can benefit from the availability of high-quality images that are always used to represent fashion items online. Some attributes cannot be extracted from the visual appearance of fashion items, such as fabric or country of origin. However, on average we believe that they do not participate greatly in the user's choice and that consequently they can be ignored for this purpose.

We wish to design a versatile method to predict joint fashionability. By versatile, we mean that a single model should be able to recommend items from any of m different categories. Furthermore, we want to be able to complete an outfit by predicting multiple items from different categories at the same time. This amounts to making predictions using various amounts of information at our disposal, a capability that we denote as robustness. This is related to multi-task learning, a subfield of machine learning that is concerned with solving multiple tasks simultaneously. In this work, independently from the problem of recommending jointly fashionable items, we also seek to demonstrate the versatility and robustness of our solution experimentally on a classic machine learning dataset.

Finally, we want to demonstrate the capability of our model to recommend jointly fashionable items using a real-world dataset of fashion images. We seek a generic solution that can be applied to any situation where jointly fashionable items must be recommended. This also includes interior design for instance, and might or might not be based on visual data. The limitations of our proposed method will be truthfully listed such that it could be compared to other approaches.

1.4 Structure

Below is a short description of each chapter and its content.

In Chapter 2, we formally describe the problem that we are trying to solve, that is to jointly fashionable items, and introduce the notation that will be used throughout this work.

In Chapter 3, we review some of the most important research domains that are related to our problem. Because our problem can be seen as a noise reduction problem, a background on noise reduction techniques will be first presented. We then introduce recommendations systems and present some of the most representative recommendation systems algorithms. Finally, we conduct a review of the literature on the specific task of recommending jointly fashionable items.

In Chapter 4, we formally define the model that will be used throughout this work, a DAE. We then provide an interesting perspective on what this kind of model is essentially doing by considering it through the scope of topology. Two characteristics of DAEs, that

we denote as robustness and versatility, are demonstrated. We outline our method and provide some arguments regarding its validity. The previously described method is then implemented, and experimental results are analyzed.

In Chapter 5, we try to solve the original problem of this work, which is to recommend jointly fashionable items. We first provide an argument as to why we believe DAEs could be used for this purpose. As in Chapter 4, we describe our method and then experiment on a fashion dataset. Experimental results are then analyzed. Finally, our solution is compared with previous works and its limitations are outlined.

We conclude by outlining the major points of this work. Several ways to improve the results we obtained are provided, as well as possible extensions to this work.

2. Problem statement

We are interested in the problem of recommending an item that is jointly fashionable to a set of other items based on their visual appearance. By saying that two or more items from different categories are jointly fashionable, we mean that they are likely to be observed together on fashionable outfits. In practice, each image would have to go through two steps that are the extraction of item regions from the image, or segmentation, and the extraction of visual features from items to obtain vectors of features. Vectors of visual features can be obtained through different means that we won't detail in this section. For simplicity, the visual feature vector of an item will be directly referred to as an item. We first define how images, and the items they contain, are generated using a simple data generation model. Then, we introduce both an inventory of items to recommend and a query containing items for which the recommendation of a jointly fashionable item is requested. Finally, we propose a formal mathematical definition for the problem at hand.

Consider an image that is i.i.d. sampled from some unknown probability distribution. Each image contains exactly m items, and each item in a given image is labeled with an element from the finite set $C = \{0, 1, ..., m-1\}$, where the label corresponds to the category of the item. We assume that no two items in an image can be labeled with the same category. Each item is defined as a column vector $\mathbf{a}^{(i)} \in \mathbb{R}^{k \times 1}$ of visual features, where $i \in C$ is the label of the item and k is the same for all items. Furthermore, item $\mathbf{a}^{(i)}$ is assumed to be i.i.d. sampled from a statistical distribution $\mathcal{D}^{(i)}$, of whom there is a total number of m for each label $i \in C$, such that $\mathcal{D}^{(i)} = proj_i(\mathcal{D})$. An image can then be represented as a matrix of items $\mathbf{A} \in \mathbb{R}^{k \times m}$, such that $\mathbf{A} = [\mathbf{a}^{(0)}, \mathbf{a}^{(1)}, ..., \mathbf{a}^{(m-1)}]$, and $\mathbf{a}^{(i)} \sim \mathcal{D}^{(i)}, \forall i \in C$.

Figure 2.1: Representation of an incomplete query $\mathbf{Q}^{\neg i}$

In this problem, we assume that items are not combined chaotically inside images, and therefore that outfits are stylish. In situations where items are apparels or home furniture, this is a reasonable assumption. If they combined chaotically, then we couldn't do better than recommending random items, and this problem would be trivial. This means that the column vectors of $\mathbf{A}$, which are random variables, are not independent of each other. It also means that, on top of considering the fashionability of standalone fashion items, the joint fashionability between fashion items should also be taken into account. Therefore, it must be possible to learn the joint distribution between the m different categories of items to provide recommendations. Note that here, item complementarity and joint fashionability are equivalent concepts.

In the problem at hand, a query must be given to the system in order to provide a recommendation. A query is an image provided by a user that is represented as a matrix of items $\mathbf{Q} = [\mathbf{q}^{(0)}, \mathbf{q}^{(1)}, ..., \mathbf{q}^{(m-1)}]$, where $\mathbf{q}^{(i)} \sim \mathcal{D}^{(i)}, \forall i \in C$: this is referred to as the query. Additionally, we also define an incomplete query $\mathbf{Q}^{\neg i}$ as a query for which a recommendation of an item from a specific label $i \in C$ is requested. $\mathbf{Q}^{\neg i}$ is such that its i 'th item is replaced with an arbitrary fixed value column vector $\mathbf{\emptyset} \in \mathbb{R}^{k \times 1}$ which symbolizes the absence of the corresponding item from the query; namely $\mathbf{Q}^{\neg i} = [\mathbf{q}^{(0)}, ..., \mathbf{q}^{(i-1)}, \mathbf{\emptyset}, \mathbf{q}^{(i+1)}, ..., \mathbf{q}^{(m-1)}]$.

To recommend an item that would complement a given query $\mathbf{Q}^{\neg i}$, we need an inventory of potential items to recommend. We define m different finite sets of items $I^{(i)} = \{\mathbf{p}_1^{(i)}, \mathbf{p}_2^{(i)}, ..., \mathbf{p}_{n_i}^{(i)}\} \sim \mathcal{D}^{(i)}, \forall i \in C$, where n_i is the number of items in inventory $I^{(i)}$. Note that, for a given instance of the problem, the content of the inventory is predetermined and fixed both in size and content across all possible queries, whereas a query $\mathbf{Q}$ can be any possible composition of items observed inside a random image. Therefore, the quality of the recommendation is constrained by the size and content of the inventories.

We can now formally introduce the objective of the problem at hand. Given an incomplete query $\mathbf{Q}^{\neg i}$, and for any label $i \in C$, we wish to recommend an item $\mathbf{p}^{(i)}$ from our inventory $I^{(i)}$ as to maximize the fashionability of item $\mathbf{p}^{(i)}$ with other items in $\mathbf{Q}^{\neg i}$. This notion of fashionability is learned from a set of observations $S = \{\mathbf{R}_j\}_{j=0}^{n_S-1}$, where each observation is an image represented as a matrix of items $\mathbf{R}_j = [\mathbf{r}_j^{(0)}, \mathbf{r}_j^{(1)}, ..., \mathbf{r}_j^{(m-1)}]$, such that $\mathbf{r}_j^{(i)} \sim \mathcal{D}^{(i)}, \forall i \in C$. Let $\dot{\mathbf{p}}^{(i)}$ be the most complementary item to the incomplete query $\mathbf{Q}^{\neg i}$ in inventory $I^{(i)}$. $\dot{\mathbf{p}}^{(i)}$ can be estimated by maximizing the joint probability density between the m different categories of items considered at $\mathbf{p}^{(i)}$ and $\mathbf{Q}^{\neg i}$ with S as a parameter. The objective can then be summarized as

$$\underset{\mathbf{p}^{(i)} \in\ I^{(i)}}{\arg\max}\ f(\ \mathbf{p}^{(i)}\ ,\ \mathbf{Q}^{\neg i}\ ;\ S\ )\ , \tag{2.1}$$

Note that in practice $f(\ \cdot\ ;\ S\ )$ is not the true joint probability density between items, but rather is a parameterized model that is set to estimate it as accurately as possible. Its parameters must be learned separately using a learning algorithm whose accuracy will depend, in part, on the number of observations n_S. By unwrapping items in $\mathbf{Q}^{\neg i}$, $f(\ \cdot\ ;\ S\ )$ can be rewritten as follow:

$$f(\ \mathbf{p}^{(i)}\ ,\ \mathbf{Q}^{\neg i}\ ;\ S\ ) = f(\ \mathbf{q}^{(0)}\ ,\ ...\ ,\ \mathbf{q}^{(i-1)}\ ,\ \mathbf{p}^{(i)}\ ,\ \mathbf{q}^{(i+1)}\ ,\ ...\ ,\ \mathbf{q}^{(m-1)}\ ;\ S\ ) \tag{2.2}$$

Furthermore, the higher the dimensionality of item vectors k and the number of categories m, the sparser observations in S will be in the input space and the greater the number of

observations will be necessary to accurately estimate the true joint distribution of items. This is also known as the curse of dimensionality when solutions in low dimensions quickly become computationally prohibitive as the dimensionality increases.

In Equation 2.1, we try to find the best candidate amongst a limited set of items in the inventory, which can be seen as a classification of an incomplete query $\mathbf{Q}^{\neg i}$ into one of the potential items of the inventory. However, as we will later see and contrary to Equation 2.1, we will rather think about this problem as a regression task and not as a classification task. For this purpose, we use a specific version of the autoencoder architecture, the DAE. We first introduce the regular autoencoder framework before presenting its denoising variant. But first, we provide an overview of the subjects that are related to our work.

3. Related Work

3.1 Noise Reduction

A user query $\mathbf{Q}^{\neg i}$ whose item with label i is missing can be seen as a noisy signal. Removing the noise from $\mathbf{Q}^{\neg i}$ translates into finding an item that is most likely to be observed alongside items in $\mathbf{Q}^{\neg i}$, i.e. the one that we should ideally be recommending. In that sense, our problem is related to noise reduction problems. In this section, we provide a short introduction to the problem of reducing noise as well as how it is done.

Noise reduction is the process of removing noise from a signal, where noise denotes an undesirable signal component. A noise is typically a random signal which can be characterized by its statistical properties. If a noise wasn't random but deterministic, then the task of removing it would be trivial. Note however that noise can also be adversarial, in which case additional precautions must be taken. In practice, it is impossible to completely remove noise from a signal, and noise reduction will only result in a slight but useful increase in the Signal-to-Noise ratio (SNR). Noise reduction is critical in signal processing, and more specifically in telecommunications, speech recognition, or image processing.

There exists a wide variety of noises; the most common ones being white and pink noise. White noise has a constant power spectral density on a linear frequency scale and is naturally approximated by the thermal noise of charge carriers, like electrons in an electrical conductor. Pink noise has a constant power spectral density on a logarithmic frequency scale, which is also known as a Bode plot. Pink noise is often observed in biological systems. For instance, it has been observed as a response of biological systems stimulated with white noise [70]. Interestingly, the human perception of sound (and light) is known to be approximated by a logarithmic scale, which is why pink noise is commonly used as a calibration signal in audio engineering. This logarithmic perception is also known as Fechner's law, which states that the perceived audio or visual stimulus is proportional to the logarithm of the stimulus intensity. Let p be the intensity of the stimulus that is perceived by an individual, Fechner's law is such that

$$p = k \ln \frac{S}{S_0} \ , \tag{3.1}$$

where k is a constant, S is the physical intensity of the stimulus, and S_0 is the minimum stimulus intensity that can be perceived by the individual.

Noise reduction can be applied to any digital or analog signal. Consider a noisy signal $\widetilde{\mathbf{x}}$ as the sum of a clean signal of interest and a noise signal, that is $\widetilde{\mathbf{x}} = \mathbf{x} + \boldsymbol{\epsilon}$. A simple step to reduce noise in $\widetilde{\mathbf{x}}$ is to model the statistical property of both $\mathbf{x}$ and $\boldsymbol{\epsilon}$. Then, according to those observed properties, the simplest approach to reducing noise is to filter some frequency bands more than others. For example, in two-dimensional representations of natural processes like photographs, the noise can often be considered to be a white Gaussian noise. White refers to the i.i.d. nature of the noise of each pixel, which allows one to consider each pixel's noise as a random variable. This noise will

tend to increase high spatial frequencies in images, as most of the original information in images is located on the lower end of the spatial frequency spectrum. Therefore, filtering out high spatial frequencies would effectively reduce noise in the image.

A basic approach to reducing high frequencies is to average the signal values over a given 2-dimensional window using a mean filter. Mathematically, this is achieved by convolving a boxcar function with the noisy signal $\widetilde{\mathbf{x}}$. However, mean filtering will tend to increase some harmonic frequencies, which is undesirable. This is because the Fourier transform of the boxcar function is the Sinc function, and its absolute value has evenly-spaced high-frequency bumps as shown in Figure 3.1. Those high-frequency terms are also known as harmonic frequencies. In that regard, a better filter is the Gaussian filter whose Fourier transform is also a Gaussian function: consequently, the amount of applied filtering gently increases along with the frequency of the noisy signal $\widetilde{\mathbf{x}}$. However, it is impossible to create an exact Gaussian filter either numerically and analogically, which is why they are often approximated.

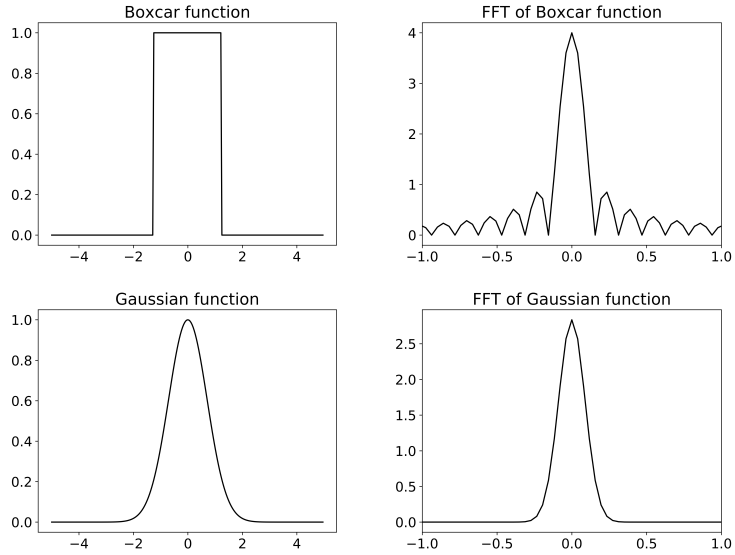

Figure 3.1: Boxcar and Gaussian functions with corresponding Fourier transform.

It must be noted that filtering out high frequencies, depending on how flat the original signal's spectrum is, will remove part of the original signal of interest as well. However, this is usually not noticeable if the cutoff frequency is high enough, as the human perception of visual signals is weakly sensitive to high frequencies. In practice, the flattest possible response in the signal band along with a steep roll-off in the stopband is often sought out. Better filters than mean or Gaussian filters are used for this purpose, like the Butterworth filter. Chebyshev filters of type I or II are also sometimes favored for their steeper roll-off in the stopband, with the drawback of being less flat in the passband.

We now provide more details on the low-pass Gaussian filter that is used to reduce the noise by filtering out high frequencies. Note that filters are also being reffered to as kernel. Considering the noisy signal as a function $f : \mathbb{R} \to \mathbb{R}$, convolving f with a Gaussian kernel is known as a Weierstrass transform, which is a functional of the function f that makes it smoother. The convolved Gaussian kernel can be defined as

$$\frac{1}{\sqrt{4\pi t}} e^{-x^2/4t} \ , \tag{3.2}$$

where t controls the smoothness of the transformation. The one-dimensional generalised Weierstrass transform $W_t[\cdot]$ of function f is

$$W_t[f](x) = \frac{1}{\sqrt{4\pi t}} \int_y f(y) e^{-\frac{(x-y)^2}{4t}} dy \ , \tag{3.3}$$

which can also be expressed as a convolution

$$W_t[f](x) = \frac{1}{\sqrt{4\pi t}} \int_y f(x-y) e^{-\frac{y^2}{4t}} dy \ . \tag{3.4}$$

Interestingly, the generalised Weierstrass transform $W_t[\cdot]$ in 3-dimensional space can be used to accurately model the diffusion of heat in materials through time. Equation 3.2 is then referred to as the heat kernel, where t becomes the time elapsed since heat diffusion began in the material. In fact, Joseph Fourier proved in 1822 that Expression 3.2 is the fundamental solution of the heat equation. Beyond physics, the generalised Weierstrass transform can also be used to make some functions easier to analyze. Indeed, it has been shown that if the Weierstrass transform of f exists between real numbers a and b, then it also exists everywhere in between. Furthermore, the Weierstrass transform generates analytical functions which are smooth by definition, hence infinitely differentiable.

The white Gaussian noise that we previously considered was static, in the sense that its statistical properties were stable over time. If the noise is dynamic, better filtering methods are available. The Kalman filter [31], by both estimating the expected signal using previous measurements and measuring the actual signal through time, can estimate the noise, or prediction error, by subtracting the two. Its performance can be tuned using the Kalman gain, which balances the importance given between current measurements and the estimation of the state of the noise and signal from previous measurements. The author Rudolf Kalman proved that the Kalman filter is the optimal linear filter if the noise is a white Gaussian noise whose parameters μ_Y and σ_Y are changing over time.

So far, the previously described noise reduction techniques only made use of the estimated statistical properties of the noise and signal. However, in most cases, we do have some prior knowledge about the signal of interest that can be leveraged. Given enough examples of noisy signals and their clean counterparts, we can train a machine-learning algorithm to denoise more effectively. For instance, convolutional DAEs have been successfully applied to noisy images [50]. Similarly, in this work, we use learning algorithms to denoise and hence predict what the original signal is given its noisy counterpart.

3.2 Recommendation systems

The problem of recommending complementary items is part of another important research domain, the one of recommendation systems. Numerous methods were designed since the advent of the internet in the 90's. The major types of recommendation systems as well as their most representative algorithm are presented in this section. A literature review of previous work on the problem of recommending complementary items is then conducted.

Recommendation systems are a specific type of information filtering system designed to predict which items in an inventory are most likely to be bought by users. They solve two problems. The first one is the discovery problem, that is the difficulty for customers to find the products they need online efficiently. For example, Amazon proposes more

than 12 million different products online, therefore helping their customers find what they need faster is a strong value proposition. The second problem that recommendation systems are solving is the one of increasing the retailer's revenue. This is why some of the largest online retailers are willing to invest millions in research and development to improve their recommendation systems.

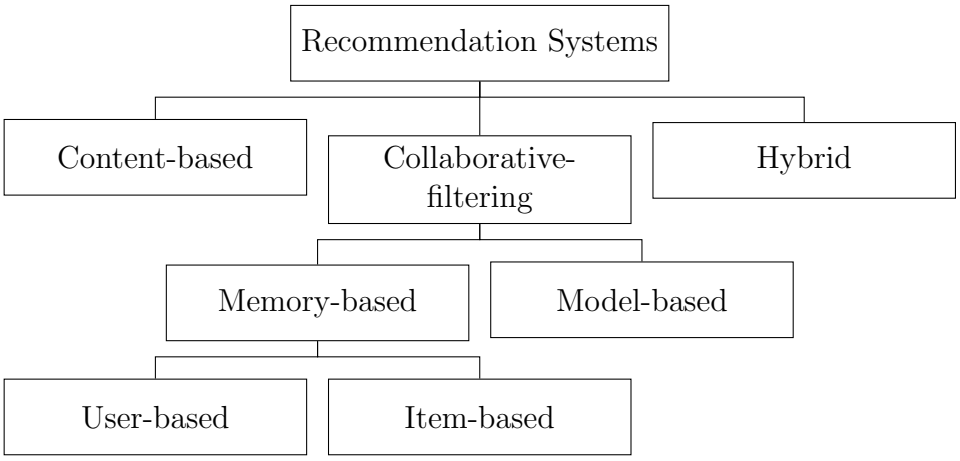

Figure 3.2:
Hierarchy of recommendation systems

Recommendation systems have been historically divided into two categories, namely collaborative-filtering (CF) and content-based filtering. Content-based recommendation systems recommend users items whose attributes are similar to those of items that they previously bought or interacted with. CF algorithms differentiate from content-based algorithms in that they do not need to understand or have access to the content of items they are recommending. They are rather based on the observed interactions between users and items. CF algorithms can be further divided into memory-based and model-based algorithms. Memory-based CF is concerned with predicting missing ratings as a function of other known ratings given by similar users (user-based) or received by similar items (item-based). Opposed to memory-based CF algorithms are model-based CF algorithms that are leveraging more advanced machine learning techniques to predict missing ratings. Finally, at the crossroads between content-based and CF recommendation systems, hybrid systems merge both approaches to get the best that each can provide [54] [4]. The relationship between those different methods is summarized in Figure 3.2 which proposes a hierarchy of recommendation systems. We now look more closely into these different types of recommendation systems and present some of their most representative algorithms.

3.2.1 Content-based

The first approaches to making recommendations were based on the item's attributes and the preferences of users for those same attributes. Attributes can include information like color, category, texture, or even visual features like Scale Invariant Feature Transform (SIFT) [51]. Content-based recommendation systems have several limitations. The first one is that items must be annotated with attributes that can be processed by computers. Nowadays, there are almost no limitations on the kind of information a computer can

process, but this has not always been the case. Until recently computers weren't able to fully understand the content of images for example. A second limitation of content-based algorithms is their limited ability to provide relevant recommendations. Let's say for example that a given user has been buying a lot of DVDs online from the movie genre *Nouvelle Vague*, a French art film movement of the late 50's. Although this user only bought DVDs, it would be most natural for them to also be interested in literature on the *Nouvelle Vague* genre, or even biographies of directors who initiated this art movement. Unfortunately, unless all items related to the *Nouvelle Vague* genre are annotated using a *Nouvelle Vague* tag, a content-based filtering algorithm won't be able to pick up this relationship and will only recommend DVDs whose attributes are similar to the one the user has previously bought. A content-based approach can also be limited by the noisy nature of item attributes, which are often manually annotated and therefore error-prone. That being said, because content-based approaches are easy to implement and provide satisfactory performance, they are still being used extensively. We now present a simple content-based algorithm used for document recommendation and search, the Term Frequency-Inverse Document Frequency (TF-IDF) algorithm [39].

TF-IDF is one of the first and most popular content-based algorithms. It is used as a measure of relevance between a term and a document that is part of a corpus of documents. Variations of TF-IDF were found in most search engines before being gradually replaced by more advanced Natural Language Processing (NLP) techniques. TF-IDF is made up of two measures, the term-frequency (TF) which measures the frequency of a term in a document, and the inverse document frequency (IDF), which measures the importance of a term in the corpus of documents by assigning it a weight that is inversely proportional to its frequency in the corpus of documents. It is often recommended to normalize the TF measure either by the maximum term frequency in the document or by the length of the document. Let $f_{t,d} \in \mathbb{N}$ be the raw count of term t in a document d, and $l_d \in \mathbb{N}^+$ be the length of document d. The length normalized TF score of t in d is:

$$TF(t, d) = f_{t,d} \ / \ l_d \tag{3.5}$$

Let D be a corpus of documents as a set, such that $d \in D$. The IDF score measures the specificity, or rarity, of a term t across all documents $d \in D$. It is traditionally defined as the logarithm of the total number of documents in the corpus D divided by the number of documents containing the term t:

$$IDF(t, D) = \log \frac{|D|}{|\{d \in D : t \in d\}|} \tag{3.6}$$

The final TF-IDF relevance score between a term t and a document d in a corpus D is

$$TFIDF(t, d, D) = TF(t, d) \times IDF(t, D) \ . \tag{3.7}$$

Note that numerous variations of TF-IDF exist, most of them being based on how the TF and IDF scores are normalized, but all are fundamentally grounded on the same principles of term frequency and term specificity. TF-IDF is known to work very well as a heuristic, but the theory behind it is not well understood. Theoretical interpretations of it have been reviewed in [61]. We now look more closely into the other major type of recommendation systems, that is collaborative-filtering.

3.2.2 Collaborative-filtering

Let's consider again our user who is interested in the *Nouvelle Vague* movie genre. By analyzing his purchase history, we can find similar users to him and realize that on top of buying DVDs on the *Nouvelle Vague* genre, those similar users also bought books on the *Nouvelle Vague*. A CF algorithm will be able to pick-up this relationship and recommend books on the *Nouvelle Vague* subject. In that regard, CF algorithms are generally considered to be superior to content-based algorithms [10]. However, a major drawback of CF algorithms is a decrease in their prediction accuracy as the sparsity of user-item observations increases. Indeed, usually, only a small subset of user-item interactions are observed. This eventually culminates with the impossibility to recommend anything if interactions between users and items are too sparse, an issue also known as the cold start problem. Hybrid recommendation systems, by integrating both CF and content-based filtering, were designed in part to solve this issue.

Memory-based CF

We now look more closely into CF. CF algorithms can be divided into two different sets of algorithms, namely memory-based and model-based algorithms. Memory-based algorithms are also known as neighbor-based and work by finding similar users, or neighbors, to the user we are interested in. It also has an item-based variant that considers similar items instead of similar users. Let's consider as an example a simple memory-based CF algorithm based on the similarity between different users (user-based). The ratings given by neighbors to a specific item are combined using an aggregation function to predict the rating the user would give to this same item. Let I_u be the set of items that user u has rated, and $r_{u,v}$ be the rating given by user u to item v. We can define the mean rating b_u given by user u as:

$$b_u = \frac{1}{|I_u|} \sum_{v \in I_u} r_{u,v} \tag{3.8}$$

The predicted rating $\hat{r}_{u,v}$ given by user u to item v is then

$$\hat{r}_{u,v} = b_u + k \sum_{u'=0}^{N} w(u,u')(r_{u',v} - b_{u'}) \ , \tag{3.9}$$

where $w(u,u')$ is a similarity metric between users u and u', N is the total number of users, and k is a normalizing factor such that $k \sum_{u'=0}^{N} w(u,u') = 1$. In practice, the similarity function $w(u,u')$ would often be a Pearson coefficient [60] or a cosine similarity [9]. Memory-based algorithms, just like content-based algorithms, are known to have scalability issues, as all observed ratings must be processed each time a prediction is requested. This makes the use of those algorithms challenging for online systems, where recommendations must be provided in real-time. Note that the prediction speed can be increased by approximating the nearest-neighbor search at the cost of lower prediction accuracy. This can be achieved either by keeping the top-N similar users to u as in [60], or by using a similarity threshold to discard users that are not similar enough.

Model-based CF

Opposed to memory-based CF algorithms are model-based CF algorithms, which try to model the underlying process that generates observed ratings. This often translates into learning what is known as latent factors. Due to its state of the art prediction accuracy and speed [10] amongst CF recommendation systems, matrix factorization (MF) methods are often favored. MF became widely known when Simon Funk discussed how good his model (dubbed Funk MF) was performing on the Netflix dataset in a blog post during the Netflix Prize competition in 2009. Inspired by the mathematical SVD decomposition, it consists of factorizing a sparse rating matrix $\mathbf{R} \in \mathbb{R}^{m \times n}$ into two sub-matrices $\mathbf{P} \in \mathbb{R}^{m \times k}$ and $\mathbf{Q} \in \mathbb{R}^{k \times n}$, also known as latent factors, such that $r_{u,v} \sim p_u q_v$, $\forall (u, v) \in \mathcal{K}$ where $\mathcal{K}$ is the set of user-item pairs for which ratings are known. If the mathematical SVD is deterministic, matrix factorization is an optimization problem, and solutions are obtained using an iterative process. A benefit that MF has over mathematical SVD is that latent factors in $\mathbf{P}$ and $\mathbf{Q}$ are forced to be non-negative, which is more practical. The biased MF variant proved to be very effective. It consists of separating the user, item, and global biases from the latent factors $\mathbf{P}$ and $\mathbf{Q}$. This simple modification led to a significant increase in prediction accuracy. Biased MF was later improved by Koren et al. with SVD++[35] which added the implicit feedback that lies behind any user-item interaction, be it rated good or bad, as a sign of interest from users. timeSVD++[36] was later introduced to model temporal changes in users' behaviors using trigonometric functions.

Let's consider as an example the biased SVD model proposed by Funk in 2009. The predicted rating $\hat{r}_{u,v}$ given by user u to item v is given by

$$\hat{r}_{u,v} = \mathbf{p}_u \mathbf{q}_v + \mu + b_u + b_v \ , \tag{3.10}$$

where μ is the global mean rating, b_u is the mean rating given by user u and b_v is the mean rating received by item v. We can express the prediction error as $e_{u,v} = r_{u,v} - \hat{r}_{u,v}$. To avoid overfitting of the training data, a regularization term weighted by the hyper-parameter $\beta \in \mathbb{R}^+$ is added to the loss as to penalize large value for the parameters, which are symptomatic of overfitting:

$$\mathcal{L} = \sum_{(u,v) \in \mathcal{K}} e_{u,v}^2 + \beta(||\mathbf{p}_u||^2 + ||\mathbf{q}_v||^2 + b_u^2 + b_v^2) \tag{3.11}$$

Latent factors $\mathbf{P}$ and $\mathbf{Q}$ are either learned using Alternative Least Square (ALS) or Stochastic Gradient Descent (SGD). ALS works by iteratively fixing $\mathbf{p}_u$ and solving for $\mathbf{q}_v$, and then conversely, for each pairs $(u, v) \in \mathcal{K}$. This gives a sequence of convex optimization problems that can be solved using the Ordinary Least Square (OLS) method, and for which a closed-form solution is known. Refer to Appendix 7.1 for more informations on the closed-form solution of OLS. Gradient descent directly computes the gradient of the non-convex loss with respect to each parameter of the model before updating them. The stochastic variant of gradient descent uses the loss of each observed rating iteratively instead of using the combined loss over all observed ratings as in Equation 3.11. Both SGD and ALS are guaranteed to converge to a local minimum. A notable difference between the two is that with ALS the error is guaranteed to either decrease or stay the same, because of the convexity of the problem, whereas an SGD iteration can either increase or decrease the error. Let's consider the SGD derivation as an example. The derivative of the loss function for a single observation (u, v) with respect to the user bias

parameter b_u is

$$\frac{\partial \mathcal{L}}{\partial b_u} = -2e_{u,v} + 2\beta b_u \tag{3.12}$$

Let $\eta \in \mathbb{R}^+$ be the learning rate which controls how big of a step we take to update the parameters. The constant factor of 2 can be merged with the learning rate η for simplicity. With SGD, the parameters of the model are updated as follow:

$$\begin{aligned} b_u &\leftarrow b_u + \eta(e_{u,v} - \beta b_u) \\ b_v &\leftarrow b_v + \eta(e_{u,v} - \beta b_v) \\ \mathbf{p}_u &\leftarrow \mathbf{p}_u + \eta(e_{u,v}\mathbf{q}_v - \beta \mathbf{p}_u) \\ \mathbf{q}_v &\leftarrow \mathbf{q}_v + \eta(e_{u,v}\mathbf{p}_u - \beta \mathbf{q}_v) \end{aligned} \tag{3.13}$$

SGD is often favored for the simplicity of its implementation, the speed at which parameters are updated, and its capacity to sometimes cross "bumps" in the loss function to escape a local minimum. ALS on the other hand provides longer but more accurate updates of the parameters. ALS also has the benefit of being easily parallelized, which allows the practitioner to train MF models on very large datasets. Parallelized versions of SGD, like DSGD [20] [40], were later proposed to address the issue of using SGD to train MF models on large datasets using a cluster of computers. Overall, a key benefit of model-based CF over memory-based CF and content-based systems is the ability of those CF models to make fast online predictions. This is at the cost of training a model beforehand which can take a non-negligeable amount of time depending on the complexity of the model and the number of parameters.

Note that MF methods are linear, in that they try to model observed ratings by assuming a linear relationship between the unknown latent factors that we try to learn. However, linear models are fairly simple and might fail to accurately model the process that generates observed ratings. This led practitioners to experiment with non-linear models.

Neural approach to CF

Lately, many neural approaches to CF have been developed with the hope that the non-linearity of those models would help to design better recommendation systems. One of the first attempts at making recommendations using neural networks was proposed by Sedhain et al. with AutoRec [65], where a DAE was used to impute missing ratings in a sparse rating matrix. It has two variants, I-AutoRec and U-AutoRec, depending on how the rating matrix is being fed to the input layer of the DAE. Indeed, one can either slice the rating matrix in the column (item) direction or the row (user) direction. While an increase in prediction accuracy compared to their baselines was observed, they acknowledged that increasing the number of layers, which allows for more non-linearity of the model, only provided modest gains in prediction accuracy. This hints that CF latent factors can be effectively learned using simple models. In 2019, realizing that both user and item perspectives of the sparse rating matrix were complementary, Zhu et al. proposed a joint CF DAE architecture and reported an improvement in the top-k recommendation score [76]. Although our work is related to [65] and [76], we aren't exactly in a CF setup and are rather concerned about a visual content-based approach to making recommendations.

Despite the small gains reported by [65] and [76], the enthusiasm for neural-based CF

algorithms hasn't faded, with a large number of increasingly complex neural architectures being proposed for this purpose, each reporting a supposedly increase in prediction accuracy. This echoes the fact that, lately, applied machine learning researchers have been mostly focussing on absolute performance metrics [72], which doesn't necessarily improve the quality of the perceived recommendation nor imply usability and practicality of the proposed solution in a real-life setting. A systematic review of neural-based CF algorithms as of 2019 was conducted in [14] and raised two problems. First, they found that research results were often difficult or even impossible to reproduce. This includes inaccessibility to the source code, unclear pre-processing steps, or an unknown initialization method for the parameters. Second, baselines were sometimes badly tuned, which is in favor of proposed methods, and their choice is questionable. By implementing what could be reproduced with reasonable efforts, they found a second issue: most proposed solutions could be outperformed by well-known, decade-old properly tuned baselines. Generally speaking, it seems that traditional CF recommendation problems are solved to a point where no significant improvements in terms of absolute prediction accuracy could be gained. They concluded their work by calling for improved scientific practices in this area.

Recommendation systems are traditionally concerned about recommending a single item at a time to users. However, one might observe that items can also be used or observed together. Beyond the simple recommendation setup that we previously described, we now look at the specific problem of recommending items that are complementary to one another, something that is independent of user interests or past behaviors.

3.2.3 Recommending for joint fashionability

Recommending for joint fashionability is the task of recommending items that are jointly fashionable with one or more other items that are provided by a user.

An item-item CF approach can be seen as the first step toward this type of recommendation. Presented by Amazon in 2003 [42], this approach is popular under the headline "Customers who bought X also bought Y". For each item in the inventory, a list of other items with which it was co-purchased is kept. Instead of building an item-item matrix, which is memory intensive and doesn't scale, they use an adjacency-list data structure to benefit from the highly sparse nature of item-item interactions. Given a currently considered item, the list of items with which it was co-purchased is filtered. In a CF setting, items can be represented as binary vectors of users, where elements set to 1 imply that users associated with the index of those elements interacted with the item. The similarity between two items can then be defined as the similarity between their two user vectors. The similarity between the user vector of an item and the user vector of every item with which it was co-purchased is computed. Co-purchased items are then sorted according to their similarity score, and the top N items are recommended, where N is the number of items to recommend. Contrary to the memory-based CF approach described in Section 3.2.2, the lists of co-purchased items can be computed offline, which allows for faster online recommendations.

However, co-purchased items do not necessarily imply complementarity of those same items, and considering so is too strong of an assumption. A well-known legend in marketing is the one of beer and diapers, which are supposedly often purchased together. Other than this potential relationship, these two items aren't related in any other way. However, gin, tonic, and lemons are clearly complementary, as they are consumed together

as part of the same recipe. For a notion of complementarity to be applicable is highly dependent on the context and type of items that are considered. Recently, the stock investing application Robinhood made a surprising usage of that kind of item-item CF recommendation system; the application recommended its users the stocks that similar users to them previously bought. The rationality behind such recommendation is highly debatable and could lead to errors of judgment from beginner investors. Beyond the absolute prediction accuracy of the recommendation, understanding the implication and relevancy of such recommendation is also important.

One of the first real attempts at recommending jointly fashionable items based on their visual appearance was proposed by Iwata et al. in 2012 [29]. They presented a Latent Dirichlet Allocation (LDA) model to match top apparel to bottom apparel and conversely. LDAs are part of the topic model family, a group of statistical generative models for discrete data that is frequently used in text-mining. They extracted top and bottom bounding boxes from photographs. Faces were located using a simple but accurate face detection algorithm [62]. Then, using common knowledge of human proportions, and knowing that their reference photographs were exclusively made of straight and camera facing persons, they deduced from the face's position and size the position of the top and bottom apparel regions. After randomly checking extracted regions, they reported that 73% of images could be considered as being correctly extracted. A total of approximately $3,300$ photographs were used. Color and Scale Invariant Feature Transform (SIFT) [51] features were extracted and concatenated into vectors. The topic model was then trained using them to learn the parameters of the Dirichlet distributions, as well as the latent topics, each latent topic being the equivalent of a fashion style. The top and bottom regions of each photograph were all encoded into low-dimensional vectors of topic proportions. Bottoms were matched to tops whose topic proportion was most similar, and conversely. One might observe that their region extraction process was suboptimal. First, the top region (and to some extent the bottom region as well) can contain one or more items, like a shirt, a scarf, or a jacket, and all of those items will be considered as a single top entity. Second, not all pixels inside the detected region are part of the actual items; in fact, most of them belong to the background whose visual features are irrelevant. Clearly, the performance they obtained could be improved by using more recent region extraction methods, like semantic segmentation neural network for instance.

In 2012, Liu et al. [44] created a recommendation system for the task of dressing for the right occasion and for recommending jointly fashionable items. They used an attribute-occasion Support Vector Machine (SVM) model for the task of dressing for the right occasion, and another attribute-attribute SVM model for the task of recommending jointly fashionable items. They manually annotated the top and bottom regions of $24,417$ images with 10 occasion categories, and 7 apparel attributes, each being assigned to one or more values. Similarly to [29], top and bottom regions were extracted, but this time using a dedicated model that was proposed in [74]. Visual features including Histogram of Oriented Gradients (HOG) and Local Binary Patterns (LBP) were automatically extracted from the top and bottom regions. Overall, positive results were obtained, and they noted the superiority of non-linear SVM models over linear SVM models. They also mentioned the importance of accurately extracting top and bottom regions as a pre-processing step, as misdetections would necessarily result in bad recommendations.

Extending the idea of matrix factorization used in collaborative filtering, in 2015 Hu et al. [26] proposed another approach to recommend jointly fashionable items. They

modeled user-item and item-item interactions by learning the latent factors of an $M+1$ order tensor, where M is the number of item categories. They simplified their problem by only considering 3 categories, namely top, bottom, and shoes. A gradient boosting method was used to map feature vectors from the feature space to a latent space. Their feature vectors included advanced versions of HOG and SIFT features. They gathered their training data from 150 users on Polyvore.com and obtained a training set of 180 positive outfits and 900 neutral outfits for each of them. The sparse tensor was factorized using functional gradient descent [52], a generalization of gradient descent that works in the generalized space of functions. A benefit that functional gradient descent has over traditional gradient descent is that a non-convex parametrized function can become convex in the function space. It is then possible for the model to avoid getting stuck inside of a local minimum. Although positive results were reported, the practicality of such a method remains to be demonstrated. Furthermore, the computational complexity induced by the functional gradient descent hasn't been discussed. All-in-all, any implementation would require a solid understanding of the complex mathematical theory that is involved.

In the fashion industry, the appearance of products is of utter importance. We believe manually annotated attributes to be in their vast majority just a compressed summary of the visual appearance of fashion apparel. It is well-known that low-level visual features like the one extracted by methods such as HOG or SIFT fail to extract the abstract content and meaning that images contain. Due to their capacity to learn complex patterns and their non-linearity, convolutional neural networks are able to fully leverage and understand the content of images. Therefore, a content-based approach based on latent features extracted from such models makes perfect sense and should theoretically provide better item representations, and therefore better recommendations.

One of the first attempts at recommending apparels using neural networks was proposed by Song et al. [68]. They proposed a content-based approach based on Bayesian Personalized Ranking (BPR). Instead of trying to optimize the error between a predicted rating and its ground-truth rating, BPR is rather concerned about the ranking of the most relevant items to recommend. Taking a sparse user-item matrix of interactions as input, and for a given user, BPR optimizes a ranking over all items for this specific user by using pairwise item preferences. BPR uses implicit interactions, like clicks on an item's page or search queries, instead of explicit ones like ratings or scores. Pairs of item preference are made of an item with whom the user implicitly interacted with, denoted as the positive item, and of an item with which the user did not interact, denoted as the negative item. The underlying assumption is that users prefer items with which they implicitly interacted over others. Finally, using Bayes' theorem, the posterior probability of a user to prefer positive items over negative items is maximized. This translates into maximizing the predicted score difference between the positive item and the negative item. It has been shown in the original paper of BPR [59] that BPR optimizes the Area Under ROC Curve (AUC) metric, a well-known machine learning metric, where the ROC curve is the plot of the true-positive rate as a function of the false-positive rate.

In 2018, Nakamura et al. [55] proposed a BiLSTM that generates a sequence of compatible items from different categories which are conditioned on the previous ones. LSTM stands for Long-Term Short-Term Memory and is a Recurrent Neural Network (RNN), a specific type of neural network that is generally used to process sequences of text. That being said, RNNs and by extension LSTMs can also be used to generate

sequences of any other kind of data, including latent representation of fashion apparels for instance. The bidirectional property of an LSTM refers to its added ability to process the data in both forward and backward directions. They used a pre-trained CNN model to extract latent representations of fashion items as a preprocessing step. Then, three different sub-networks were used: a Visual Semantic Embedding (VSE) network, used to join visual features and manually annotated attributes, a Style Embedding network (SE) whose task is to learn a style embedding from the whole outfit, and the BiLSTM model per se that is generating the sequence of items to recommend. Their work is based on a previous BiLSTM implementation for sequence of complementary items prediction [22]. Their contribution compared to [22] is to add the SE sub-module, which allows for fine-grained control of the generated outfit's style. The downside of using an LSTM is that it requires multiple inferences for each item that must be generated, and that based on the previous sequence of items. In an online recommendation setting, a single inference is preferable.

In 2019, Lin et al. proposed a variational model that recommends new items based on a picture of what needs to be complemented and a textual description of the requested item [41]. More specifically, a visual representation of the predicted ideal item is generated. To avoid the generation of blurry images, an issue that is often observed when variational models are used along with Deconvolutional neural networks (DCNN), they separate the generative process in two stages. The first stage is a classic DCNN which generates a low-resolution image. The second stage is made of a super-resolution residual network (SRResnet) [38], which generates high-resolution images. Their model jointly learns a top encoder, a bottom encoder, and a top/bottom image generator. They advertized the superiority of this approach over the use of feature extractors made of pre-trained CNN models and reported a state of the art prediction accuracy over previous methods. Because their generator allows for the generation of either top or bottom items, we believe that this property could be extended to generate items from an arbitrary number of item categories, hence bringing the versatility property that we are seeking in this work.

As in [68] and [22], we propose to directly extract visual features from fashion apparel using a pre-trained CNN. We now present the model that will be used throughout this work, the denoising autoencoder (DAE), and show how it could be used to recommend jointly fashionable items.

4. DAEs as flexible prediction models

While applied machine learning researchers are often focused on absolute prediction accuracy [72], they often discard other traits such as practicality, flexibility, and resilience under degraded conditions. This contrasts with real-world implementations which are often made of less performant but more practical methods. We those goals in mind, we experiment with the use of Denoising Autoencoders (DAE). We first introduce this particular architecture of neural networks, and provide an interesting perspective of their inner working. Two of their characteristics are then investigated, namely their robustness, and versatility. We describe our method for measuring these two qualities, before training a DAE on Abalone, a classical machine learning dataset. Finally, experimental results are analyzed, and the relationship between DAEs and multi-task learning is discussed.

4.1 Denoising Autoencoders

DAEs are a variant of the autoencoder framework defined in Appendix 7.2.3. Instead of taking a full vector as input to extract an embedding of lower dimensionality, like regular autoencoders, DAEs are rather provided with corrupted inputs and are given the task of reconstructing a complete and uncorrupted version of their inputs on their output layer as accurately as possible.

DAEs were initially used as predictive models, and not as feature extractors [67]. Along with convolutional layers, DAEs were later used extensively in image processing tasks to reduce the noise in images [50]. Convolutional layers can learn spatial patterns from the input data, contrary to fully connected layers which, as later demonstrated in Proof 5.2, are insensitive to the spatial ordering of their input values. Convolutional DAEs have also been used in super-resolution tasks [38], where the goal is to increase the definition of an image by inferring the value of missing pixels. It must be noted, however, that in image super-resolution tasks, there is no need to keep missing entries in the input layer, as their location is fixed and deterministic. If the position of missing entries is not known in advance, which is generally the case, then missing entries must be kept. In [50], Mao et al. showed the superiority of convolutional DAEs over traditional upscaling techniques like linear and cubic interpolation or even Gaussian smoothing, as well as for noise removal. Others have used DAEs to increase the Signal to Noise Ratio (SNR) of sonar images [33] or to restore and colorize old video footages [57]. Apart from being used as predictive models, a recent work by Vincent et al. in 2008 [71] demonstrated that randomly corrupting the input of DAEs, and eventually values in every subsequent layer, could also be used to help the model learn more robust representations, which ultimately led to better accuracy on classification tasks. In our case, we used DAEs as prediction models, and not to learn useful features. We now present a vanilla DAE and discuss its specificities.

If not otherwise stated, everything that applies to the autoencoder framework defined in Appendix 7.2.3 also applies to the DAE framework. Just like a regular autoencoder,

a DAE is made of an encoding function $f : \mathbb{R}^k \to \mathbb{R}^{k'}$ and a decoding function $g : \mathbb{R}^{k'} \to \mathbb{R}^k$, respectively parametrized by the set of parameters ϕ and ψ. Contrary to regular autoencoders, it is not necessary to apply regularization, as DAEs cannot learn the identity function: indeed, the model is forced to reconstruct on its output layer a piece of information that is different (because it is corrupted) than that of its input layer. Therefore, corrupting the input layer, and eventually, other subsequent layers, acts as a form of regularization, which prevents the model from overfitting and helps it generalize to unseen observations.

Inputs are corrupted using a corruption process. We define a corruption process as a function of both the input to be corrupted and of a corruption mask. Later in this work, we will define our own corruption process. Apart from containing some kind of randomness, there is no restriction on the nature of a corruption process. If it wasn't partially random, then the DAE could learn a deterministic mapping of corrupted observations to their corresponding uncorrupted versions: in other words, it would overfit the training set and fail to generalize to unseen observations. Let $S = \{\mathbf{x}_j\}_{j=0}^{n_S-1}$ be a set of n_S observations, where each observation $\mathbf{x}_j \in \mathbb{R}^k$ is i.i.d sampled from some unknown probability distribution $q(X)$. Corrupted observations are obtained using a corruption process $\widetilde{\mathbf{x}}_j \sim q_{\mathcal{D}}(\widetilde{\mathbf{x}}_j|\mathbf{x}_j)$. The encoder and decoder functions are respectively defined as

$$\begin{aligned} \mathbf{y}_j &= f(\widetilde{\mathbf{x}}_j; \phi) \ , \\ \hat{\mathbf{x}}_j &= g(\mathbf{y}_j; \psi) \ , \end{aligned} \tag{4.1}$$

where $\hat{\mathbf{x}}_j \in \mathbb{R}^k$ is the predicted observation associated with the corrupted observation $\widetilde{\mathbf{x}}_j$, and $\mathbf{y}_j \in \mathbb{R}^{k'}$ is known as the embedding or latent vector. A simple 2-layer DAE is shown in Figure 4.1. Similar to regular autoencoders, the sets of parameters ϕ and ψ are updated through one or more epochs of Stochastic Gradient Descent (SGD) and backpropagation. In the case of a DAE, a training epoch can be generically defined as follow:

$$\phi, \psi \leftarrow \arg\min_{\phi,\psi} \frac{1}{n_S} \sum_{j=0}^{n_S-1} \mathcal{L}(\mathbf{x}_j, \ g(f(\widetilde{\mathbf{x}}_j; \psi); \phi)) \ , \tag{4.2}$$

We say generically because in practice finding ϕ and ψ as to minimize equation 4.2 is not trivial when more than two layers are involved, and because equation 4.2 doesn't tell us exactly how this is achieved. One can refer to the SGD and backpropagation techniques defined in Appendix 7.2.2 for more information on how to learn the model's parameters ϕ and ψ. Finally, $\mathcal{L}$ is some arbitrary loss function whose choice is outside the scope of this section. In the DAE framework, it will generally be a discrepancy measure between uncorrupted observations $\mathbf{x}_j$ and their corresponding predictions $\hat{\mathbf{x}}_j$ such that $\mathcal{L} : \mathbb{R}^k \times \mathbb{R}^k \to \mathbb{R}$. Equation 4.2 can be expressed in more probabilistic terms. Let q^0 be the empirical distribution associated with our set of training observations S. We are trying to learn the joint distribution

$$q^0(X, \widetilde{X}, Y) = q^0(X) q_{\mathcal{D}}(\widetilde{X}|X) \delta_{f(\widetilde{X};\theta)}(Y) \tag{4.3}$$

where $\delta_u(v)$ is the Kronecker delta function which puts mass 0 when $u \neq v$: this is defined as to show that Y is a deterministic function of $\widetilde{X}$. The objective function 4.2 then becomes

$$\phi^*, \psi^* = \arg\min_{\phi,\psi} E_{q^0(X,\widetilde{X})} \Big[\mathcal{L}(X, \ g(f(\widetilde{X}; \psi); \phi)) \Big] \tag{4.4}$$

A particularity of DAEs regarding the loss $\mathcal{L}$ is that it can be processed in different ways; every predicted value on the output layer might or might not be taken into account in the computation of the loss $\mathcal{L}$. Indeed, inputs are sometimes already corrupted: this is the case in AutoRec [65] for example, as only a fraction of user-item ratings is observed. In this case, only known ground-truth values and their corresponding predicted elements in $\hat{\mathbf{x}}_j$ are considered when computing the loss. Note that this can only be done if the corruption process is such that not all values of observations are corrupted; else, the loss would be null as all values would be ignored. On the other hand, if inputs are uncorrupted from the start, then we usually willingly corrupt them while keeping their ground-truths on the side. This is the case in [71] for instance.

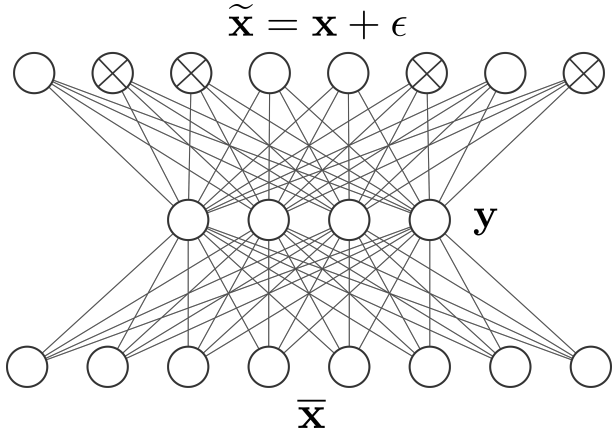

Figure 4.1: Representation of a vanilla 2-layer DAE

The capabilities of DAEs can be easily demonstrated on the MNIST dataset. Inspired from the code published on GitHub [6], we trained a DAE to remove a white Gaussian noise that is added on top of images of handwritten digits. We use MNIST, a popular dataset of handwritten digits with size 28×28 pixels, each having a single 8 bit grayscale channel. The dataset is separated into a training set of $60,000$ images and a testing set of $10,000$ images. We initialized a 4-layer DAE (2 for the encoder, 2 for the decoder) made of fully connected layers, with an embedding size of 64. Despite the dataset being made of images, no convolutional layers were used, as the simplicity and relatively low dimensionality of the data makes it unnecessary. However, for more complex images, convolutional layers are recommended. Images are normalized in the $[0, 1]$ range and rounded to their closest bound as to convert them from grayscale to black and white pixels. Pixels are then corrupted using a white Gaussian noise with distribution $\mathcal{N}(0, 0.5)$. We also flattened observations as to go from a dimensionality in $\mathbb{R}^{28\times28}$ to $\mathbb{R}^{784}$, as this simplifies both notation and implementation. MNIST can therefore be represented as a dataset $S = \{\mathbf{x}_j\}_{j=0}^{n_S-1}$ of observations, where each vector $\mathbf{x}_j \in \{0, 1\}^{784}$ is a vectorized representation of image with index j.

Because images are originally black and white, we obtain a binary classification problem for each pixel of each image. For this purpose, a binary cross-entropy (BCE) loss is used. In the information theoretical sense, given two distributions p and q defined on the same sample space Σ, the cross-entropy measures the average number of bits necessary to identify an event drawn from Σ if the coding scheme is optimized for p instead of q. There exists several such optimized coding scheme, of which the most popular and used one is the Huffman coding scheme. It is related to the Kullback-Leibler Divergence

$D(p, q)$, which measures the amount of information that is lost (in bits) when q is used to approximate p. The cross-entropy between p and q can be generically defined as:

$$\begin{aligned} H(p, q) &= H(p) + D(p, q) \\ &= E_p[-\log_2(q)] \end{aligned} \tag{4.5}$$

Because the cross-entropy loss considers vectors of probabilities only, we must make sure that the output of our model is bound in the range $[0, 1]$. For this purpose, we use the logistic function. We define as $\sigma(\hat{\mathbf{x}}) \in \mathbb{R}^{784}$ the probability vector associated with a raw prediction $\hat{\mathbf{x}}$, such that each element $\sigma(\hat{\mathbf{x}})_l$ with index l of vector $\sigma(\hat{\mathbf{x}})$ is equal to:

$$\sigma(\hat{\mathbf{x}})_l = \frac{1}{1 + e^{\hat{x}_l}} \ \forall l \in [0, k-1] \tag{4.6}$$

The general expression of the cross entropy defined in equation 4.5 can now be expressed in discrete terms for the binary classification task at hand. The binary cross-entropy of the model's predicted probability element $\sigma(\hat{\mathbf{x}})_l$ with respect to its ground-truth x_l becomes

$$H(x_l, \sigma(\hat{\mathbf{x}})_l) = -x_l \log \sigma(\hat{\mathbf{x}})_l + (1 - x_l) \log(1 - \sigma(\hat{\mathbf{x}})_l) \tag{4.7}$$

A visualisation of the training process can be seen in figure 4.2. Random observations from the testing set are displayed before being corrupted, after being corrupted, and after their corrupted versions were denoised by the DAE.

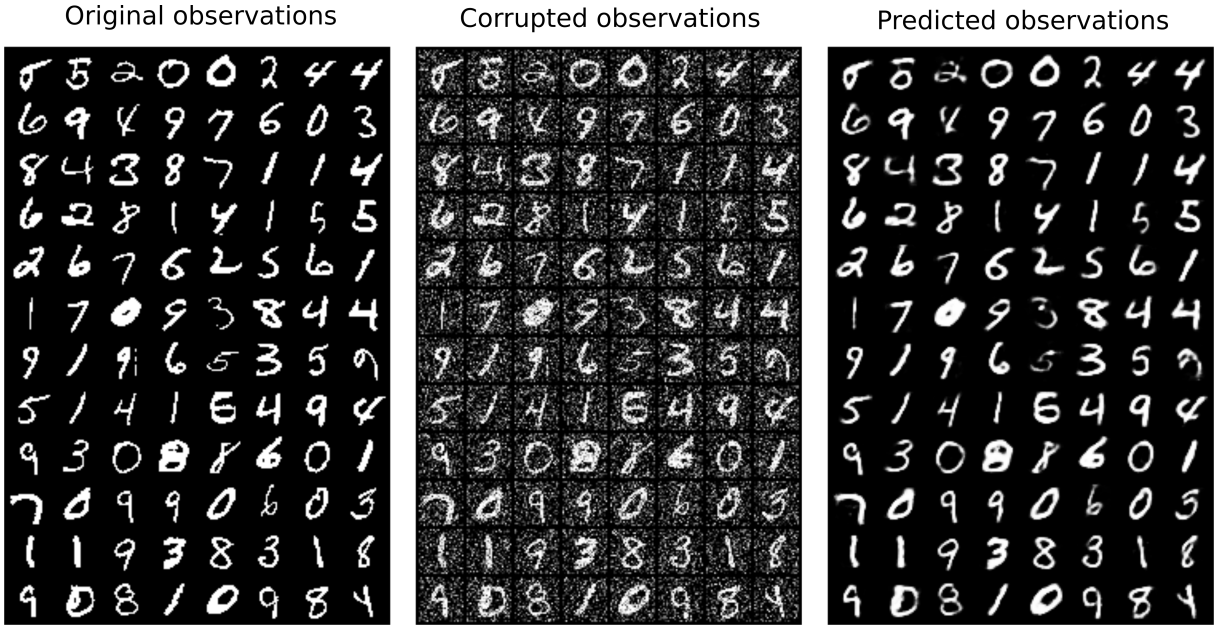

Figure 4.2: Subset of prediction results on MNIST at epoch 190/200.

If the corruption process used to corrupt the MNIST dataset was a summed white Gaussian noise, many other corruption processes can be used. In subsequent parts of this work, we experiment with what we call a binary corruption process and a block binary corruption process. To better understand the effect of noise and how DAEs denoise their inputs, we now propose an interesting perspective using a seemingly unrelated mathematical domain: the one of topological spaces and manifolds.

4.2 Manifold learning perspective

DAEs can be seen through multiples lenses, but the manifold perspective is of particular interest when it comes to understanding how and why this architecture works. One might wonder why a corrupted observation $\widetilde{\mathbf{x}}$ would be magically denoised by a DAE, and why its corresponding denoised observation $\hat{\mathbf{x}}$ can be considered a good prediction of the real uncorrupted observation $\mathbf{x}$. In [71], Vincent et al. proposed an interesting perspective on what a DAE is doing when mapping corrupted observations to their uncorrupted counterparts: they argued that DAEs are effectively learning a low-dimensional manifold near which uncorrupted observations are gathering. Inference can then be seen as the projection of a point associated with a corrupted observation in the input space onto this lower-dimensional embedded manifold. In [2], Alain et al. proposed a thorough theoretical analysis of this phenomenon.

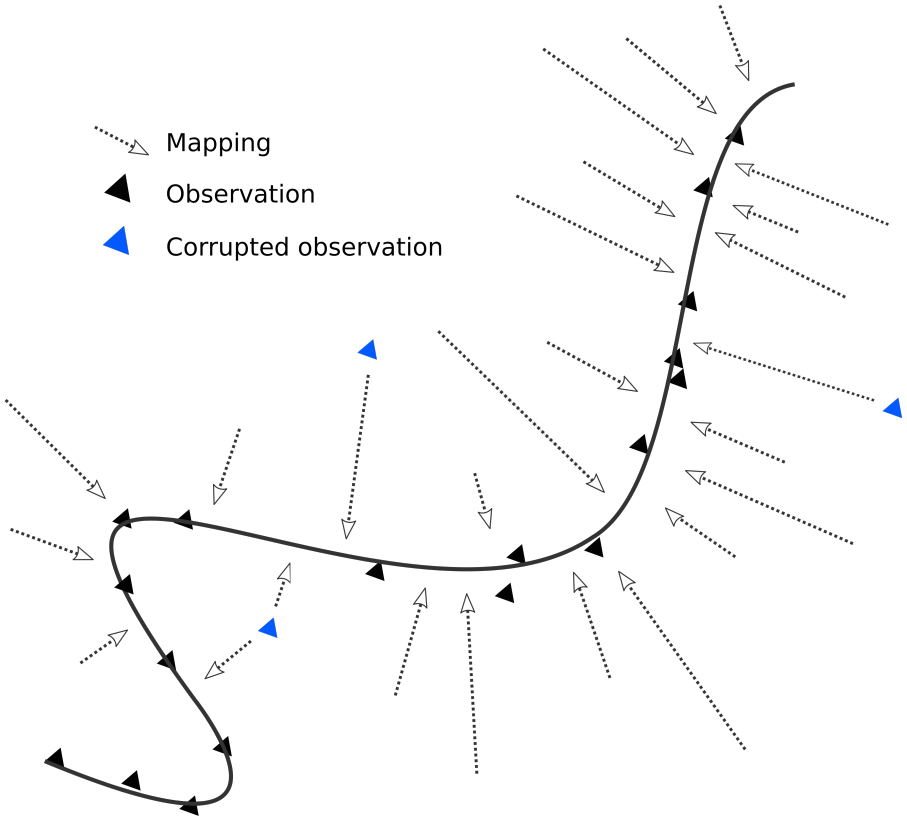

Figure 4.3:
Manifold learning perspective:
2-manifold to 1-manifold mapping

Formally, a n-manifold is a topological space with n dimensions, where the space around a point is locally homeomorphic to $\mathbb{R}^n$. For example, a sphere is said to be a 2-manifold, whereas a circle is said to be a 1-manifold. Statistically speaking, the probability for a corrupted observation $\widetilde{\mathbf{x}}$ to be far from the learned manifold is higher than that of its uncorrupted counterpart $\mathbf{x}$. Figure 4.3 proposes a representation of this manifold learning perspective inspired by a figure from [71]: the input space is $2-$dimensional, while the embedded manifold is 1-dimensional. The more complex and curved this embedded manifold is, the more expressivity will be required from the DAE to fit it accurately. This expressivity usually translates into a greater number of layers. Drawing parallels

with machine learning, overfitting in DAEs can be seen as learning a manifold that tries to pass through all training observations by being overly curvy and complex. Building on this manifold learning perspective, Sonoda et al. [69] argued that both the input and output space of a neural network can be seen as being embedded into a single high-dimensional space. They showed that the trajectory of any point on the input manifold to the output manifold forms a geometric object that is independent of parameterization, which are the weights and biases of the network, and therefore is a universal character of neural networks.

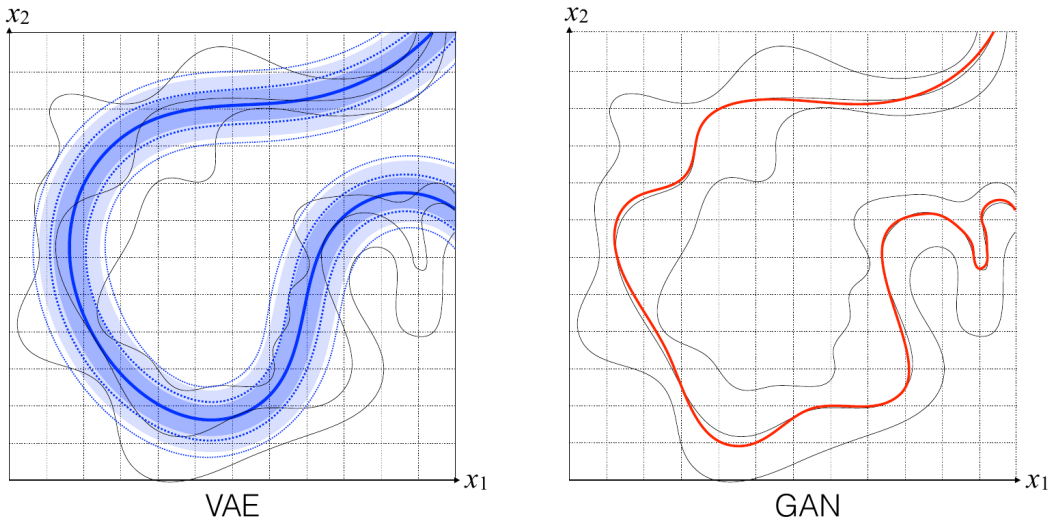

Figure 4.4: Manifold learning comparison between VAEs and GANs.

In Figure 4.3, the two mapping arrows pointing at the first corrupted observation (first blue triangle starting from the left) suggest that this corrupted observation could have different mapping trajectories. Consequently, it might let one think that a DAE could provide different predictions for a given corrupted observation. In fact, DAEs are deterministic in that they are predicting the mean of the (estimated) conditional distribution. However, a variant of autoencoders known as variational autoencoders (VAE) can be used to learn not just the mean of the conditional, but the whole conditional distribution [34]. This variational property was soon expanded to DAEs to obtain what is known as a variational DAE (VDAE) [28]. By sampling the learned conditional distributions, a VDAE could be used to recommend multiple good complementary items to a given query, which would then make this model behave like a full-blown stylist with variety: we believe VDAEs to have great potential for the task of recommending for joint fashionability.

If VAEs (and by extension VDAEs) can sample the whole estimated conditional distribution, they also present some serious limitations, as a specific family of statistical distribution must be selected to regularize the embedding layer. Standard normal distributions are often used, but might fail to fit the true distribution of features in the lower-dimensional embedded manifold [18] [17]. Generative Adversarial Network (GAN) [21], which just like VAEs are generative models, do not have this restriction and can estimate any arbitrary statistical distribution given enough expressivity (i.e. breadth and depth of the network). In Figure 4.4, which is extracted from a presentation given by Aaron Courville for the MILA institute [18], a simplification of this reality is showed.

The embedded manifold near which observations are gathering is represented by a curvy line, the embedded manifold, that is surrounded by two other curvy lines symbolizing the dispersion of observations near the embedded manifold. On the left, we can see that the VAE grossly approximates the embedded manifold whereas, on the right, the GAN is fitting it accurately. Unfortunately, GANs lack a usable inference mechanism to obtain embeddings from observations and are mostly designed to generate new unseen observations. To both approximate any arbitrary statistical distribution and allow for an embedding inference mechanism, an Adversarial Autoencoder architecture was proposed by Makhzani et al. [49].

Now that the inner working of DAEs has been demystified, we present two of the characteristics that set them apart from other models.

4.3 Robustness and Versatility

In machine learning, we commonly deal with datasets, i.e. sets of data. The data is assumed to be sampled from some unknown probability distribution that we would like to learn. Datasets are made of observations, and each observation is made of a set of one or more variables. Depending on which of those variables we end up trying to predict given others, they will either be referred to as predictor variables, which are always provided, or target variables. Machine learning can be summarized as the task of learning the statistical relationships that exist between those variables over the whole dataset, such that the knowledge we gained from it could be generalized, or verified, on unseen data that is drawn from the same data generating distribution as the original dataset. ML models are typically designed and trained to predict a very specific set of target variables using the information provided by predictor variables. But this is not the case with DAEs, which as defined in Section 4.1 do not distinguish between predictor variables and target variables. This is because DAEs process whole observations and not just the predictor variables they contain. For DAEs, any of the variables contained in an observation can be potentially missing. In fact, DAEs have several advantages over other models, of which two are of particular interest.

The first advantage of DAEs is their ability to handle the absence of any of their input variables and predict what they should most likely be given other non-missing variables. This is what we commonly refer to as *versatility*. In contrast, traditional ML models typically expect and must be trained with very specific predictor and target variables. In the DAE framework, and the autoencoder framework in general, the whole information is processed, and all variables are considered as both predictor and target variables. This completely alleviates the need to have an ML model designed and trained for every combination of potential predictor and target variables. Take for example the problem of replacing soccer athletes throughout the game. A soccer team has eleven players dispatched on the football pitch, and eleven different "roles". Although all are athletes, they do not have the same kind of attributes and their performance is not evaluated in the same way depending on their specific roles. Some athletes perform better if put alongside other athletes with the same style of play for example. Considering that any of these eleven players might need to be replaced, we would have to design and train eleven different models, where each of those models would predict the best player on the bench for a specific role to replace another player on the football pitch. On the other side, a single DAE can handle those eleven different cases. This capacity is in fact not specific

to DAEs and is related to a subfield of machine learning that is known as multi-task learning. Multi-task learning will be further discussed later in this work.

Another benefit of DAEs is their ability to provide recommendations using various levels of information at their disposal. This is what we denote as *robustness*. Providing various amounts of information to a DAE is in fact exactly like applying various amounts of corruption to observations before feeding them to this same DAE. Consider again the problem of recommending substitute players during a soccer game. What if we do not have any historical data about a forward that is already deployed on the football pitch, but still should predict which of the potential defender on the bench would best complement the currently deployed athletes? We would have to make the best prediction given the profile of athletes on the football pitch except for the forward that we do not know anything about, i.e. less information than usual. Furthermore, we might need to predict several variables at the same time. What if two players crashed into one another, resulting in severe injuries for both? Both would have to be replaced, which would require to predict which two players on the bench would best fit with other athletes on the football pitch. Preparing for this eventuality would require to train $C_{11}^2 = 55$ different prediction models, where C_n^k denotes the binomial coefficient for "n choose k". This is impractical and tedious. A single DAE, if trained correctly, could handle all of these different cases, and make a prediction given the available information at its disposal. Note that robustness is stronger than versatility, as versatility only refers to the ability to predict any single potentially missing variable out of the m different ones, whereas with robustness several of the m variables could be missing (and corrupted) at the same time. We now present our method for measuring the robustness and versatility of DAEs.

4.4 Method

In any machine learning dataset, variables can be represented as being either categorical or continuous. Categorical variables require a specific encoding before being fed into a neural network. The simplest type of encoding is to associate a natural number to each potential label. This is well suited if variables are ordinal, but will degrade the prediction accuracy otherwise. Another popular method is one-hot encoding, where a binary vector has elements for each unique label, and where the element associated with the class label is set to 1 and every other to 0. For instance, the one-hot encoded representation (also known as indicator vector in mathematics) of a categorical variable that is equal to 1 and whose potential labels are $C = \{0, 1, 2\}$ has as a one-hot vectorized representation $[0, 1, 0]$. If $|C|$ is very large, embeddings can be extracted as a preprocessing step to obtain much denser yet meaningful representations of categorical variables. However, this is at the expense of training an additional model. In our case, all categorical variables are simply one-hot encoded.

Before being fed into a DAE, we represent each observation as a concatenation of its variables, resulting into vectors in $\mathbb{R}^{k_{obsv}}$, where k_{obsv} is the dimensionality of the concatenated vector. Note that we always have $k_{obsv} \geq m$, m being the number of variables, as the dimensionality of one-hot encoded categorical variables is always greater than or equal to 1, and the dimensionality of continuous variables remains the same after being concatenated. For simplicity, in this section, we assume that the dimensionality k of all continuous variables is the same, and that all categorical variables have k different labels. Therefore, we have that $k_{obsv} = mk$. Consequently, an observation can be represented as

a matrix $\mathbf{V} \in \mathbb{R}^{k \times m}$, where $\mathbf{V} = [\mathbf{v}^{(0)}, \mathbf{v}^{(1)}, ..., \mathbf{v}^{(m-1)}]$, and $\mathbf{v}^{(i)} \in \mathbb{R}^{k \times 1}$, $\forall i \in [0, m-1]$. The concatenation of an observation's variables into a single vector $\mathbf{x}$, sometimes known as the $vec(\cdot)$ operator in matrix theory, can be formally defined as follows:

Definition 1 (Concatenation of variables.)**.** *Let $\boldsymbol{V} \in \mathbb{R}^{k \times m}$ be an observation as a matrix of variables such that $\boldsymbol{V} = [\boldsymbol{v}^{(0)}, \boldsymbol{v}^{(1)}, ..., \boldsymbol{v}^{(m-1)}]$, where each variable is a column vector $\boldsymbol{v}^{(i)} \in \mathbb{R}^{k \times 1}$. The concatenation of all variables in $\boldsymbol{V}$ is a unique vector $\boldsymbol{x} \in \mathbb{R}^{k_{obsv}}$, such that $x_{(i \times k)+l} = v_l^{(i)}$, $\forall i \in [0, m-1]$, $\forall l \in [0, k-1]$.*

Recall that, in the DAE framework, observations are first corrupted using a corruption process to obtain corrupted observations, which are then being fed into a DAE to be denoised. In the MNIST example provided in Section 4.1, the corruption process was made of a white Gaussian noise as a vector $\boldsymbol{\epsilon} \sim \mathcal{N}(0, 1)$ which was added to a clean uncorrupted input $\mathbf{x}$, i.e. $\widetilde{\mathbf{x}} = \mathbf{x} + \boldsymbol{\epsilon}$. Here, we want to completely predict missing variables, and not just remove the additive noise. Assume for simplicity that $k = 1$ for all variables. To completely predict missing variables, we use a random binary corruption mask sampled from a Bernoulli distribution, that is $\boldsymbol{\epsilon} \in Bernoulli(k_{obsv}, \lambda)$, $0 < \lambda < 1$, where λ is the probability for an element of $\boldsymbol{\epsilon}$ to be set to zero. A corrupted observation $\widetilde{\mathbf{x}}$ is then obtained by taking the Hadamard product of the uncorrupted observation with $\boldsymbol{\epsilon}$. In this case, the noise is not additive, but selectively destructive. Formally, a binary corruption process is defined as follow:

Definition 2 (Binary corruption process.)**.** *Let $C = \{0, 1, ..., m-1\}$ be the set of indices of each of the m variables, and $\boldsymbol{x} \in \mathbb{R}^{k_{obsv}}$ be a concatenated vector of variables where $k = 1$ for all variables. Let $D \subset C$ be a subset of indices that are to be corrupted, where $|D| \leq m - 1$ as at least one variable must remain uncorrupted. A binary corruption mask $\boldsymbol{\epsilon} \in \mathbb{R}^{k_{obsv}}$ of $\boldsymbol{x}$ is such that $\epsilon_i = 0, \forall i \in D$, and $\epsilon_l = 1, \forall l \in \{C - D\}$. The binary corruption process applied to $\boldsymbol{x}$ is then the Hadamard product of $\boldsymbol{x}$ with $\boldsymbol{\epsilon}$; that is $\widetilde{\boldsymbol{x}} = \boldsymbol{x} \odot \boldsymbol{\epsilon}$.*

One might question the use of the value zero to corrupt observations, and not any other value. This is an interesting point to address. First observe that a DAE, just like any autoencoder, is fundamentally a Multi-Layer Perceptron (MLP). For more information on MLPs, refer to Appendix 7.2.1. Because of the bias term of each layer, an MLP can learn how to offset any fixed value that is different from zero to zero. In practice, zero is used because inputs are usually normalized with mean of or close to 0, which reduces the probability of obtaining an exploding gradient during training and keep values in the working range of activation functions. But what about more complex corruption processes? We need a corruption process that is such that corrupted elements of the corrupted observation would not be correlated with their original uncorrupted values. Otherwise, the DAE would simply learn how to remove the noise as in Section 4.1 instead of fully predicting the variables that we are interested in. This is easily achieved by replacing the original values of variables that are to be corrupted by the outcome of a random variable that wouldn't contain any information, in the information-theoretical sense, which is the case for any fixed random variable. Let's consider the linear correlation between two random variables, which is also known as the Pearson correlation coefficient. Let A and B be two random variables, where A is the original value that is set to be corrupted, and B is the fixed value with which any outcome of A is replaced. The Pearson correlation coefficient between A and B is defined as $\rho_{A,B} = \mathbb{E}[(A - \mu_A)(B - \mu_B)] \ / \ (\sigma_A \sigma_B)$, which is clearly equal to 0 as $B - \mu_B = 0$.

As implied by Definition 2, one or more variables might be set to be corrupted. The number of potential corruption masks is clearly a function of the maximum number of corrupted variables and, in theory, we should train our DAE with every potential corruption mask applied to every observations of the training set. This begs the question: how many of those masks are there for a given maximum number of corrupted variables? Let $\delta_{max} \in \mathbb{N}^+$, $0 < \delta_{max} < m$, be the maximum number of variables that could be corrupted on any given observation, and C_m^δ be the binomial coefficient that gives the number of subsets of size δ amongst m elements. The total number of corruption masks is:

$$\sum_{\delta=1}^{\delta_{max}} C_m^\delta = \sum_{\delta=1}^{\delta_{max}} \frac{m!}{\delta!(m-\delta)!} \tag{4.8}$$

At most, we have that $\delta_{max} = m - 1$, which gives a DAE that would be trained with any potentially missing number of corrupted variables. In this case, we have that

$$\sum_{\delta=1}^{m-1} C_m^\delta = 2^m - C_m^0 - C_m^m = 2^m - 2 \ , \tag{4.9}$$

which is exponential in the number of variables per observation m. This drastically increases the number of potential corrupted observations we could train our model with, and therefore the training time. For instance, the total number of corruption masks per maximum number of corrupted variables δ_{max} can be seen in Figure 4.5 for $m = 9$. No closed form solution exists for the exact number of corruption masks $\forall \delta_{max}$, where $1 < \delta_{max} < m - 1$, but an upper bound can be found. If we consider a given fixed δ_{max} and as $m \to \infty$, we have that

$$\frac{C_m^{\delta_{max}} + C_m^{\delta_{max}-1} + C_m^{\delta_{max}-2} + ...}{C_m^{\delta_{max}}} = 1 + \frac{\delta_{max}}{m - \delta_{max} + 1} + \frac{\delta_{max}(\delta_{max} - 1)}{(m - \delta_{max} + 1)(m - \delta_{max} + 2)} + ... \tag{4.10}$$

The right side can be bound from above using the following geometric series

$$1 + \frac{\delta_{max}}{m - \delta_{max} + 1} + \left(\frac{\delta_{max}}{m - \delta_{max} + 1}\right)^2 + ... \ , \tag{4.11}$$

which is equal to $\frac{m-(\delta_{max}-1)}{m-(\delta_{max}-2)}$. Therefore, we have

$$\sum_{\delta=1}^{\delta_{max}} C_m^\delta \ \leq \ C_m^{\delta_{max}} \frac{m - (\delta_{max} - 1)}{m - (\delta_{max} - 2)} \tag{4.12}$$

As in Pascal's triangle, the binomial coefficient C_m^δ as a function of δ forms a triangle: this means that if $\delta_{max} > 1$, some quantity of applied corruption will be more likely than others, which is undesirable as the DAE could become better at denoising strongly corrupted inputs than lightly corrupted inputs. A simple two-stage process can be used to make it equally likely to apply any amount of corruption: we first randomly select the number of variables δ that are to be corrupted, and then randomly select a corruption mask amongst the potential corruption masks associated with δ.

When it comes to computing the prediction error of our model, we need to distinguish between two types of prediction error: indeed, the output of a DAE consists of

both a reconstruction of uncorrupted input variables and a prediction of the corrupted variables. The first error is obtained by only selecting the elements of the output layer whose corresponding elements on the input layer are corrupted: we define it as the partial error. The second error is obtained by taking the whole output layer for computing the error, which we call the full error. Intuitively, the partial error is representative of the model's capacity to correctly predict corrupted variables, whereas the full error tells us more about the overall capacity of the model to reconstruct its input on its output layer.

δ_{max}	1	2	3	4	5	6	7	8
#masks	9	45	129	255	381	465	501	510

Figure 4.5: Number of corruption masks per δ_{max} with $m = 9$.

We first trained our model with $\delta_{max} = 1$ and kept track of both the partial and full error. Then, to test the ability of our model to make predictions with various amounts of information at its disposal, we iteratively increased the number of corrupted variables δ and kept track of both the partial and full errors. We now provide the details of our implementation.

4.5 Implementation

We trained a DAE on the Abalone dataset, a classical machine learning dataset [56]. The original objective of the Abalone dataset is to learn to predict the age of abalone from physical measurements, like the weight or size of the shell. The age of abalone is equal to the number of rings that can be seen by cutting the shell through the cone, staining it, and observing it through a microscope. As this process is time-consuming, it invites for a faster and easier inference process.

The Abalone dataset is made of $4,177$ observations, each of them having 8 predictor variables to predict the age of abalone. As per the DAE framework, we treat the target variable like any other predictor variable, leading to a total count of 9 variables. All variables are continuous except for the sex variable, which is one of $\{Male, Female, Infant\}$; after one-hot encoding, the sex variable resulted in a vector of size 3. All continuous variables are scalars. Therefore, the concatenation of the 9 variables of each observation resulted in vectors of size $k_{obsv} = 11$.

	sex	length	diameter	height	whole_w	shucked_w	viscera_w	shell_w	rings
count	4177	4177.000000	4177.000000	4177.000000	4177.000000	4177.000000	4177.000000	4177.000000	4177.000000
mean	NaN	0.523992	0.407881	0.139516	0.828742	0.359367	0.180594	0.238831	9.933684
std	NaN	0.120093	0.099240	0.041827	0.490389	0.221963	0.109614	0.139203	3.224169
min	NaN	0.075000	0.055000	0.000000	0.002000	0.001000	0.000500	0.001500	1.000000
25%	NaN	0.450000	0.350000	0.115000	0.441500	0.186000	0.093500	0.130000	8.000000
50%	NaN	0.545000	0.425000	0.140000	0.799500	0.336000	0.171000	0.234000	9.000000
75%	NaN	0.615000	0.480000	0.165000	1.153000	0.502000	0.253000	0.329000	11.000000
max	NaN	0.815000	0.650000	1.130000	2.825500	1.488000	0.760000	1.005000	29.000000

Figure 4.6: Statistics on each variable of the Abalone dataset

Because we might want to predict both categorical and continuous variables, a specific loss function must be crafted. In its simplest form, it is made of the sum of each variable's error, which can be eventually weighted using a vector $\mathbf{w} \in \mathbb{R}^m$ which becomes a

hyperparameter of the DAE. In practice, we only scaled-down the loss of the categorical variable sex by a factor of 0.5, as its error was the only one being significantly larger (twice as large) compare to others. In the implementation section, we explain why it is sometimes necessary to weight the error of each variable to compute the combined error. Note that combining multiple errors is sensible for as long as the resulting combined loss function remains differentiable, which is the case when multiple differentiable errors are summed.

For continuous variables, we used a Root Mean Square Error (RMSE), instead of the traditional Mean Absolute Error (MAE). The MAE measures the average absolute deviation of predictions with respect to their ground-truths. The RMSE, on the other hand, is the square root of another well known metric, the Mean Square Error (MSE). One of the main differences between the RMSE and the MAE is that the RMSE penalizes more large errors compare to small errors. This also makes the RMSE sensitive to outliers, although those can be removed as a preprocessing step. Contrary to the MAE, the RMSE loss will tend to be greater whenever the sample size increases. This should be kept in mind when different RMSE results computed on different sample sizes are compared. Generally, the RMSE (and MSE) should be considered when the distribution of the continuous variable's values are Gaussian, and the MAE otherwise. Consider a vector of continuous values $\mathbf{z} \in \mathbb{R}^k$ that we are trying to predict. The RMSE between the predicted vector $\hat{\mathbf{z}} \in \mathbb{R}^k$ and its ground-truth $\mathbf{z}$ is:

$$RMSE(\mathbf{z}, \hat{\mathbf{z}}) = \sqrt{\frac{1}{k}\sum_{l=0}^{k-1}(\hat{z}_l - z_l)^2} \tag{4.13}$$

For categorical variables, we use again the cross-entropy, which is the optimal loss function under the inference framework of maximum likelihood [66]. The binary cross-entropy that was defined in Section 4.1 can be generalized to a multi-class classification problem. In that case, not only should raw predicted values be bound between 0 and 1, but they also should sum up to 1. This is achieved using a softmax function instead of the logistic function defined in Equation 4.6. Let $C = \{0, 1, ..., k-1\}$ be the set of labels for a multi-class classification task, and $\mathbf{z} \in \mathbb{R}^k$ be a ground-truth vector that is one-hot encoded. Our model predicts a raw vector $\hat{\mathbf{z}} \in \mathbb{R}^k$ whose elements correspond to the predicted weight for each of the k potential labels in C. The softmax function applied to $\hat{\mathbf{z}}$ gives a predicted categorical distribution vector $\sigma(\hat{\mathbf{z}}) \in \mathbb{R}^k$ whose elements $\sigma(\hat{\mathbf{z}})_l$ are equal to

$$\sigma(\hat{\mathbf{z}})_l = \frac{e^{\hat{z}_l}}{\sum_{l'=0}^{k-1} e^{\hat{z}_{l'}}} \quad \text{such that} \quad \sum_{l=0}^{k-1} \sigma(\hat{\mathbf{z}})_l = 1 \ , \tag{4.14}$$

where $\sigma(\hat{\mathbf{z}})_l$ denotes the l'th element of vector $\sigma(\hat{\mathbf{z}})$. The multi-class cross-entropy loss between a predicted categorical distribution vector $\sigma(\hat{\mathbf{z}})$ and its ground-truth one-hot encoded vector $\mathbf{z}$ can then be defined as

$$CE(\sigma(\hat{\mathbf{z}}), \mathbf{z}) = -\sum_{l=0}^{k-1} z_l . \log_2(\sigma(\hat{\mathbf{z}})_l) \tag{4.15}$$

In practice, because our ground-truth is one-hot encoded and all probability is given to a single label $i \in C$, the class, we can rewrite this expression as to ignore the incorrect labels. Let $\pi_1(\mathbf{z})$ be the index i of $\mathbf{z}$ such that $z_i = 1$ and $z_l = 0, \forall l \in C - \{i\}$. The

optimized multiclass cross-entropy of $\sigma(\hat{\mathbf{z}})$ with respect to its ground-truth $\mathbf{z}$ becomes

$$CE(\sigma(\hat{\mathbf{z}}), \mathbf{z}) = -\log_2(\sigma(\hat{\mathbf{z}})_{\pi_1(\mathbf{z})}) \tag{4.16}$$

This optimization highlights the fact that the multiclass cross-entropy loss only rewards or penalizes depending on the predicted probability of the correct label, the class, and not the incorrect ones.

All software was implemented using Python3.7. Python has become the standard language for machine learning and AI applications. It is both simple to use and powerful. Initially known as a scripting language, it slowly became a fully-fledged high-level programming language that allows for fast development iterations. We used Pytorch to define and train our model, an open-source AI framework developed by Facebook and based on Torch. Torch, while no longer being maintained, is a machine learning library, a scientific computing framework, as well as a scripting language based on the Lua programming language. Pytorch is one of the two most popular AI framework with TensorFlow, another open-source project from Google. Those two AI frameworks differ drastically in how they define the computation that they apply to the data. Both generate a computational graph, which is a Directed Acyclic Graph (DAG). A DAG is a graph with no cycle. TensorFlow's computational graph is statically defined beforehand, whereas Pytorch's computational graph is generated dynamically at runtime from a previously defined model and can be modified on the fly, which is much more flexible. TensorFlow is often favored in a production environment, in part because it is supported natively by Google's Tensor Processing Unit (TPU), an ASIC dedicated to neural network computations. Pytorch is more often used in an academic environment for its ease of use and flexibility. To easily reproduce our software environment, we used Conda, an environment and package management system which allows us to quickly install, update, and distribute Python software environments. Our workstation is made of Debian 10 Linux operating system with CUDA 10 and an NVIDIA 1080ti GPU.

During the training of the previously described model, we encountered exploding gradients. This is undesirable as it halts the training process. Exploding or vanishing gradients are recurrent issues when training deep neural networks. This a consequence of the sequence of non-linear transformations that are successively applied to the input data: the deeper the model, the higher the probability of obtaining them. Although our model only had 4 layers, because we trained with $\delta_{max} > 1$, the norm of the gradients varied greatly and couldn't be kept in a usable range using only the learning rate and regularization weight. Indeed, the higher the number of corrupted variables, the greater the error, and consequently the greater the norm of the gradients. A simple solution is to divide each gradient by its norm as to normalize the norm of all gradients. However, by doing so we might overshoot a local minimum of the loss function as it gets closer. Usually, as the model gets closer to a local minimum, the gradient will get closer and closer to zero. Another solution, which we used, is to clip the gradients by a constant $c \in \mathbb{R}$, such that every gradient norm that is above c will be set to c. This preserves the ability to "slow down" as we approach a local minimum of the loss function. Correctly setting both the training rate and the clipping value c allows us to trade a lower probability of exploding gradients for slightly longer training time, as the size of the steps taken is limited.

As each of the m input variables could be corrupted, we obtained a total number of $9 \times 4177 = 37,593$ corrupted observations to train with. 70% of the dataset was used for

training, 20% for validation, and 10% for testing. We kept track of the full and partial errors. No significant gain could be observed from increasing the depth of the network, which is why we stuck to 4 layers (2 for the encoder, 2 for the decoder). The weights of the neural network were initialized using the He method that is defined in Appendix 7.2.2, and its biases were all set to zero. The training rate was set to 0.00005 and the regularization weight to 0.000001. The model was trained for 100 epochs using batches of 64 observations.

4.6 Results

We now analyze some of the experimental results we obtained on the Abalone dataset. As explained in Section 4.4, we used two different types of errors; the full error and the partial error. The full error can be seen as a reconstruction error, as all elements of the output layer, except for those related to the corrupted variables, are provided by the input layer. The partial error however is a pure prediction error, as it only considers corrupted and therefore missing variables on the input layer. Recall that here the full and partial errors are in fact the sum of each variable's error, which depending on the variable's type will either be a RMSE or a multi-class cross-entropy.

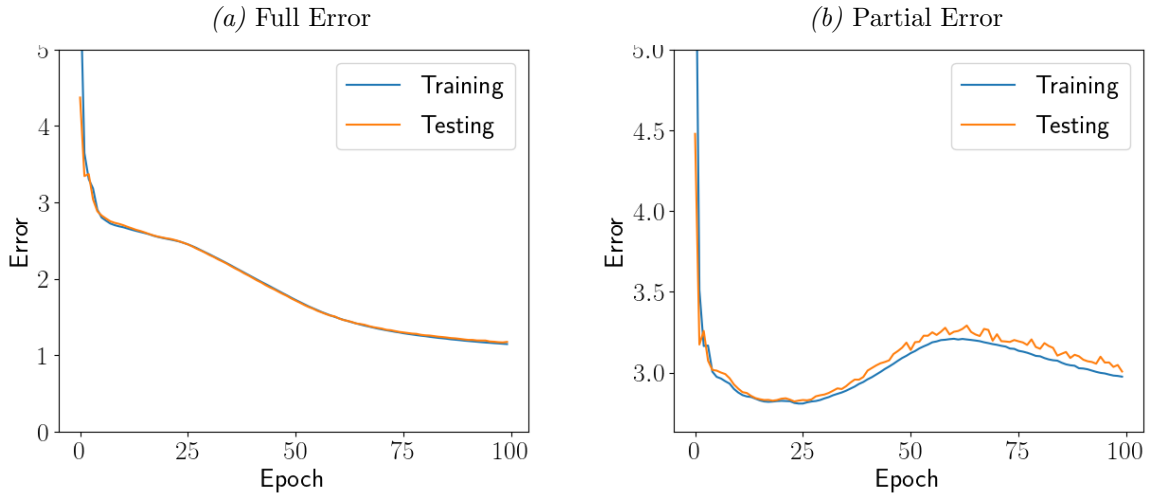

Figure 4.7: Full and Partial Error on the Abalone dataset, $\delta_{max} = 1$

We first trained the DAE on Abalone with $\delta_{max} = 1$. Figures 4.7a and 4.7b respectively display the full and partial error of the DAE. Let's first consider the full error of the DAE displayed in Figure 4.7a. We can see that both the full training and testing errors are decreasing. This informs us that the model does learn to reconstruct its input on its output layer accurately, like any regular autoencoder. Now, consider the partial error displayed in Figure 4.7b. Both the training and testing partial errors got down to reach a minimum and then increased again. Ideally, training should be stopped whenever the training and testing errors start to increase again. This second figure shows that the model can predict corrupted variables. Because the error is made of the sum of each corrupted variables' error, which can be any of the m different variables, it is also in favor of the claim DAEs are versatile. Versatility refers to their ability to handle the absence

of any of their input variables and to predict what they should most likely be given other uncorrupted variables. We did not display the partial error of each variable as in the Abalone dataset only the age of abalone variable is supposed to be predicted. However, by comparing the partial error of the age of abalone, which is an RMSE error, we observed that it was slightly higher than what other machine learning practitioners obtained, with an RMSE of 2.57 on average, although the RMSE of this variable sometimes got down to 2.15 which is close to the best performance that others obtained. In the next section, we provide a potential reason behind this sometimes sub-optimal prediction accuracy.

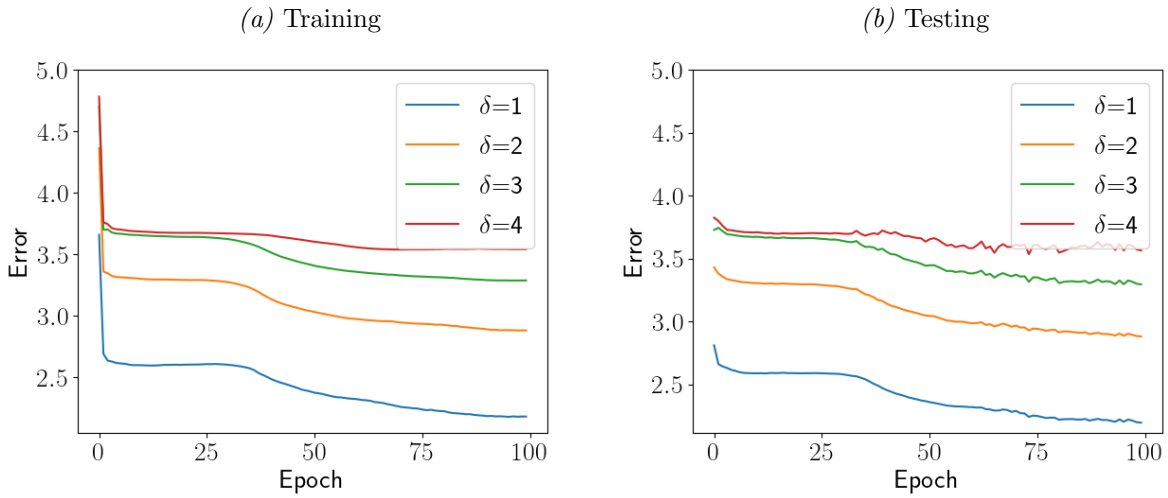

Figure 4.8: Partial Error per $\delta \in \{1, 2, 3, 4\}$ on the Abalone dataset, $\delta_{max} = 4$

We now consider different amounts of applied corruption by training the DAE with $\delta_{max} = 4$. Figure 4.8 displays the partial error for each $\delta \in \{1, 2, 3, 4\}$. We set $\delta \leq 4$ to limit the number of potential corruption masks to use during training. Indeed, a large δ_{max} drastically increases the training time. The relationship between the number of corruption masks and δ_{max} can be seen in Figure 4.5. We can see in Figure 4.8 that both the training and testing partial errors are gradually increasing as the number of corrupted variables δ increases. Again, those experimental results are in favor of the claim that DAEs are robust, as the DAE seems to make the best possible prediction given the available information at their disposal. Consequently, Figures 4.8a and 4.8b also show the capacity of DAEs to predict multiple variables at the same time, as corrupting δ variables on the input layer also mean that δ variables are predicted on the output layer.

One can observe that both the training and testing errors in Figure 4.7b quickly converged toward a minimum. A potential reason for this is that making predictions from the Abalone dataset does not require a lot of non-linearity from the model. Indeed, Abalone is a simple dataset and very satisfying results can be obtained using simple linear models [56]. This highlights the fact that deep neural networks, although able to approximate any arbitrary function, are sometimes unnecessary for simple prediction tasks. Note that the complexity of a neural network is a direct function of its number of layers and that it is entirely possible to design a 2-layer DAE without activation functions, which would then result in a linear model.

One might wonder why the testing error is sometimes equal or lower than the training error and argue that it should be the opposite. During a training epoch, we go over all

training observations in batches. We apply a forward pass to each batch, compute the loss, and then differentiate the loss with respect to each parameter of the model before updating those same parameters. Therefore, as we iterate through the training batches, our model gets better and better. When the testing error is computed, the new weights are used, whereas the training error is the combined error of each training batches that were computed using less optimized weights. This is why the testing error is sometimes lower than the previously computed training error, especially during the first epochs when large gradient steps are taken. A simple solution would be to recompute the training error at the end of the epoch with the newly optimized parameters, but this is at the expense of additional computations.

Now that both the robustness and versatility of DAEs have been demonstrated, we discuss some of the limitations of DAEs.

4.7 Limitations

A limitation that we quantified in Section 4.4 is the exponential increase in the number of potential corruption masks as the maximum number of corrupted variables δ_{max} increases, and which implies an exponential training time. However, there is another limitation that acts on prediction accuracy and that is related to the ability of DAEs to predict multiple variables at the same time.

The two qualities that we previously investigated, robustness, and versatility, are related to multi-task learning. Multi-task learning is the task of learning to solve multiple tasks at the same time using a single model. In practice, this amounts to making multiple predictions from the same input data. From this perspective, DAEs are multi-task learners. The assumption behind multi-task learning is that the knowledge gained on a specific task overlaps with the knowledge necessary to solve another related task. Multi-task learning is also related to transfer learning, where the parameters of a trained model set to solve a given task are used to initialize another model that is supposed to solve another task. This can be used in dynamic environments to quickly adapt to new situations for example. However, multi-task learning has some limitations that DAEs also share due to their multi-task nature.

It has been shown that the knowledge necessary to solve two different tasks can overlap, leading to the notion of task relatedness [11]. When tasks are not related, the model must balance its efforts between multiple objectives at the same time, which leads to a suboptimal prediction accuracy on some of the tasks. The relatedness of different tasks with respect to some shared dataset can be either learned or assumed a priori [37]. In practice, multi-task learning with neural networks often involves learning a single joint embedding space, supplemented by specific output modules dedicated to each of the tasks at hand. By output modules, we denote a small neural network whose output participates in solving a single task and not others, contrary to the joint embedding part of the network whose output feeds all output modules. It has been observed that the less related a set of tasks are, the bigger their specific modules must be to compensate. In [47], a greedy approach to learning the optimal output module size has been proposed. Another issue in multi-task learning is when the prediction error of a given task is consistently larger than others and dominates others. It leads to an overall set of parameters in the model that will benefit the prediction accuracy of this dominant task. Scaling, normalizing, or clipping gradients are potential ways to solve this issue.

In the model that we previously presented, we had a task per input variable, which is m different tasks. Those tasks are clearly related to each other but might not be 100% related. Our model did not contain any specific output modules for each of the different tasks at hand. Although we did witness a prediction accuracy that was sometimes close to what other machine learning practitioners obtained on the Abalone dataset (more specifically, for the age of abalone variable), we did not formally compare DAEs against other traditional models. We believe that adding sub-modules specific to each task (i.e. variables that we are trying to either reconstruct or predict on the output layer) could improve both the full and partial errors.

5. Co-occurence learning using DAEs

We now experiment with the use of DAEs to recommend jointly fashionable items. In this chapter, we first provide an argument as to why we believe DAEs are well suited for this task and introduce the main differences in our approach compared to Chapter 4. We then define our method, that is how we prepared the input data, the nature of our corruption process, a proof of its validity, and how we measured the performance of the proposed model. Details of our implementation are provided and results are analyzed. Finally, we compare our method with previous work and outline some of its limitations.

5.1 Joint fashionability implies Correlation

In Chapter 4, we used a classical machine learning dataset to investigate two qualities of DAEs that we denote as their robustness and versatility. The variables of each observation were first concatenated into single vectors and then corrupted using the binary corruption process defined in Definition 2, before being fed into a DAE. We apply the same principle to recommend jointly fashionable items with some necessary modifications. In this section, we talk about those differences and explain why we believe DAEs can be used to recommend jointly fashionable items.

When one talks about fashionability, it is assumed that someone, or something, is dictating what is fashionable and what is not. Let's consider a set of fashion outfits (i.e. a set of observations) specifically selected by fashion stylists. For simplicity, we assume that each outfit is made of m different apparels, and each has a different label $i \in C$. If we trust the judgment of those fashion stylists, then we can also consider that apparels in each of those outfits have some degree of positive fashionability. In fact, part of a good style is to have a strong positive joint fashionability between the items that are composing it. As previously stated, we will generically refer to apparels as items, and will in fact always refer to their corresponding feature vectors that we previously extracted using a feature extractor. Learning the notion of fashionability that exists between items of different categories in this selected set of outfits effectively translates into learning the correlation that exists amongst the feature vectors that are used to represent each of those apparel. In that sense, when considering a given set of outfits, the human notion of joint fashionability that is assumed necessarily implies a mathematical correlation of the feature vectors. This is good news since machine learning, and by extension, neural networks and DAEs, are made for the specific purpose of learning the correlation that exists amongst different variables to make predictions. This is why we believe DAEs, with their characteristics of robustness and versatility that we previously demonstrated, are well suited for the task of recommending jointly fashionable items. From a set of selected outfits, we extract the feature vector of all apparels and then estimate the joint empirical distribution of feature vectors for each of the m different categories.

Compared to the model that we trained on the Abalone dataset detailed in Chapter 4, there are some important differences in the way we must design our solution. A

first difference is that the variables we consider are all dense feature vectors, that is vectors of continuous values, with the same dimensionality in $\mathbb{R}^k$, where $k > 1$. Feature vectors representing each of the m different items in each observation are concatenated into vectors of size $k_{obsv} = km$ as defined in Definition 1. Note that feature vectors can be obtained through different means but must encode the characteristics of items as accurately as possible while remaining dense representations. Indeed, increasing the dimensionality of feature vectors would increase the sparsity of observations in the input space, which would consequently make it harder for a machine learning model to learn from the input data. Later, in the implementation section, we explain how feature vectors are obtained and later further elaborate on the potential limitations of this important preprocessing step.

A second difference, which is a direct consequence of input vectors being made of m concatenated vectors and not scalars, is that we must use a specific corruption process. It consists of erasing the sequence of values related to a particular item and not just randomly selected standalone values of the input vector, which is what the binary corruption process defined in Definition 2 was doing. We call it a block corruption process, which is the result of block corruption masks being applied to observations. Indeed, if we want to predict an item, then we must remove all values related to this particular item in a corrupted observation. This block corruption process presents the same limitations as the binary corruption process defined in Definition 2, i.e. an exponential increase in the number of potential corruption masks as the maximum number of corrupted variable δ_{max} increases. In the next section, a formal definition of this corruption process is provided. Note that this type of corruption process is not entirely new, and has been used along with convolutional DAEs and GANs to predict missing contiguous pixels in images [3] [43]. However, to the best of our knowledge, it has never been used to corrupt sequences of non-visual data.

Finally, recall that we consider the problem of recommending jointly fashionable items as a regression problem and not as a classification problem. This translates into predicting the ideal items whose joint fashionability with the user query is maximum using a DAE and not directly the best item $\cdot\mathbf{p}^{(i)}$ in our limited inventory $I^{(i)}$. Therefore, we match the predicted ideal item with its most similar counterpart in $I^{(i)}$ by conducting a nearest-neighbor search. Consequently, the quality of the recommendation will be bound by the content and size of our inventories of items. A metric that is more than just a function of the reconstruction or prediction errors is necessary to fully reflect the quality of the recommendation. In the next section, a relative ranking loss will be proposed to solve this issue.

5.2 Method

Our corruption process consists of setting the contiguous sequence of k values related to an item with some label $i \in C$ to zero, as opposed to randomly setting a set of values to zero using a binary corruption process. More specifically, label i is a random variable sampled from a discrete uniform distribution, i.e. $i \sim \mathcal{U}\{0, m-1\}$, where $\mathcal{U}\{\cdot\}$ is a discrete uniform distribution and its support is the set of labels C. Therefore, from the model's perspective, each of the m items in an observation have an equal probability of being corrupted. Note that, given a query $\mathbf{Q} = [\mathbf{q}^{(0)}, \mathbf{q}^{(1)}, ..., \mathbf{q}^{(m-1)}]$ with corresponding concatenated vector $\mathbf{x}$, there exists exactly m associated corrupted input vectors, as any

of the m different items $\mathbf{q}^{(i)}$, which are column vectors of $\mathbf{Q}$, can be corrupted. In practice, during training and for each training epoch, we make sure that the m potential corrupted observations associated with a given observation will be provided to the model exactly once. Therefore, observing a given observation whose item with label i is corrupted will increase the probability of observing this same observation with another item with label $i' \in C - \{i\}$ being corrupted, and put zero mass on the previously observed corrupted observation: this is like drawing balls from a jar without replacement. From the model's perspective however, this effect becomes negligible as $n_S \to \infty$. We now formally define the block corruption process that we use by extending Definition 2 from Section 4.4.

Definition 3 (Block corruption process)**.** *Let $\boldsymbol{Q} = [\boldsymbol{q}^{(0)}, \boldsymbol{q}^{(1)}, ..., \boldsymbol{q}^{(m-1)}]$, $\boldsymbol{Q} \in \mathbb{R}^{k \times m}$, be a query as a matrix of item vectors $\boldsymbol{q}^{(i)} \in \mathbb{R}^{k \times 1}$ with corresponding concatenated vector $\boldsymbol{x} \in \mathbb{R}^{km}$. A block corruption mask $\boldsymbol{\epsilon}^{\neg i} \in \mathbb{R}^{km}$, for some label $i \sim \mathcal{U}\{0, m-1\}$, is such that values $\epsilon^{\neg i}_{(i \times k)+l} = 0$, $\forall l \in [0, k-1]$, and $\epsilon^{\neg i}_{(j \times k)+l} = 1$, $\forall j \in C - \{i\}$, $\forall l \in [0, k-1]$. The block corruption process applied to $\boldsymbol{x}$ is then the hadamard product of $\boldsymbol{x}$ with $\boldsymbol{\epsilon}^{\neg i}$ for some label $i \in C$; that is $\boldsymbol{x}^{\neg i} = \boldsymbol{x} \odot \boldsymbol{\epsilon}^{\neg i}$.*

One might wonder if replacing values $x^{\neg i}_{(i \times k)}$, $x^{\neg i}_{(i \times k)+1}, \dots, x^{\neg i}_{(i \times k)+(k-1)}$ in $\mathbf{x}^{\neg i}$ with a sequence of zeros is a valid kind of corruption process for a DAE. In the Autorec model [65], we have a corruption process where the noise vector is drawn from a Bernouilli distribution $Bernouilli(\lambda)^{k_{obsv}}$, where k_{obsv} is the dimensionality of $\mathbf{x}$ and λ, $0 < \lambda < 1$, is the probability for any value of vector $\mathbf{x}$ to be set to zero. Although in [65] the rating matrix is not corrupted since most user-item ratings are unknown from the start, we can consider their rating matrix to be the result of the binary corruption process that we defined in Definition 2. We argue that our corruption process is in fact similar to the one used in Autorec [65].

Theorem 1. *The corruption process that consists in randomly setting some values of a row or column slice vector of a rating matrix to zero is equivalent to the corruption process that consists in setting a contiguous sequence of those same values to zero, given an appropriate permutation of the rows or columns of the rating matrix, as such permutation does not fundamentally change the information of this same rating matrix.*

Proof. Let $\mathbf{A} \in \mathbb{R}^{d \times d'}$ be a sparse rating matrix, such that $a_{u,v} = 0$ for some column index u and some row index v with probability λ, where $0 < \lambda < 1$. We define $\mathbf{a}_{u,*}$ and $\mathbf{a}_{*,v}$ as vectors representing a slice of matrix $\mathbf{A}$ in the column and row directions respectively, for some column index u and user index v. Consider that there is no specific order to the rows of $\mathbf{A}$, which can be permuted at will, as this would not fundamentally change the information contained in $\mathbf{A}$ because permutations are bijections. This also applies to the columns of $\mathbf{A}$, but not to both as a rating relates one specific column (item) entry to another specific row (user) entry. There is always a way to permute the columns of $\mathbf{A}$ such that missing values in any $\mathbf{a}_{u,*}$ become contiguous, and conversely for $\mathbf{a}_{*,v}$ by permuting the rows of $\mathbf{A}$. Therefore, randomly setting several values of any $\mathbf{a}_{u,*}$ or any $\mathbf{a}_{*,v}$ to zero is equivalent to setting a contiguous sequence of those same values to zero in a matrix $\mathbf{A}'$ whose columns or rows have been correctly permuted. □

If our corruption process is equivalent to the corruption process used in Autorec [65], it remains to be proven that applying a permutation to the input of a DAE wouldn't change its behavior. DAEs, just like autoencoders, are fundamentally made of stacked layers of fully connected layers. This is also known as a multilayer perceptron (MLP).

For more information on MLPs, one can refer to Appendix 7.2.1. We argue that MLPs are not impacted by any permutation that might be applied to their input, and more generally to the spatial ordering of their input values.

Theorem 2. *Multi-Layer Perceptrons (MLP) are not sensitive to the spatial ordering of their input values.*

Proof. Consider the first layer of an MLP, which is a fully connected layer. It's output $\mathbf{y} \in \mathbb{R}^d$ is defined as $\mathbf{y} = \sigma(\mathbf{W}\mathbf{x} + \mathbf{b})$, where $\mathbf{W} \in \mathbb{R}^{d' \times d}$ and $\mathbf{b} \in \mathbb{R}^{d'}$ respectively denote the learned weights and biases, $\mathbf{x} \in \mathbb{R}^d$ is the input, and $\sigma : \mathbb{R}^{d'} \to \mathbb{R}^{d'}$ is a partially differentiable activation function. In a fully connected layer, every neuron of the input layer is connected to every neuron of the next layer, which is why $\mathbf{W}$ is not sparse. Therefore, for any permutation of values in the input $\mathbf{x}$, the same permutation can be applied to the rows of $\mathbf{W}$ to obtain the same previous output $\mathbf{y}$. It follows that, for any permutation applied to $\mathbf{x}$, and for a fixed output $\mathbf{y}$, $\exists \mathbf{W} \in \mathbb{R}^{d' \times d}$ such that the previously defined perceptron equation remains true. We conclude that MLPs are not sensitive to the spatial ordering of their input values. □

We now present the metrics we used to measure the performance of our solution. As in Chapter 4, we have a dataset of observations $S = \{\mathbf{x}_j\}_{j=0}^{n_S-1}$, where each observation is the result of the concatenation of the m feature vectors that represent each of the m items in the observation. Because all observations are made of vectors of continuous values, we use the RMSE defined in equation 4.13 for the same reasons as in Chapter 4. We measure both the training and testing RMSE between uncorrupted observations $\mathbf{x}_j$ and predicted observations $\hat{\mathbf{x}}_j$. In each of those two cases, we further differentiate between the full RMSE, which compares all elements of $\mathbf{x}_j$ against all elements of $\hat{\mathbf{x}}_j$, and the partial RMSE which only takes into account elements of $\mathbf{x}_j$ and $\hat{\mathbf{x}}_j$ related to the corrupted item with label i. Furthermore, as we already demonstrated the capability of DAEs to predict multiple potentially missing items at the same time in Chapter 4, we set $\delta_{max} = 1$.

In addition to the reconstruction error of the DAE, we measure the quality of the recommendation with respect to our limited inventory by crafting a ranking error. We call it the Relative Inventory Ranking Error (RIRE). The idea is the following. First, consider a predicted ideal item as well as its corresponding ground-truth item, both with category $i \in C$. The ground-truth item is part of an inventory containing all items of the same category i. We compute the similarity of the ideal predicted item with each item in this inventory and then sort all items in the inventory according to their similarity score. If our model's performance is good, then the ground-truth item will be ranked first because our prediction will be close to its ground-truth, and consequently the similarity score between our prediction and its corresponding ground-truth item will be high. The converse also applies if our prediction is bad, leading to the ground-truth item being ranked not last but, on average, randomly. Note that this method implies to build a set of temporary inventories from the testing set of observations and that those temporary inventories are different from the one that will be used to make recommendations in real life.

Let $T = \{\mathbf{V}_j\}_{j=0}^{n_v}$ be a testing set of observations, where each matrix $\mathbf{V}_j = [\mathbf{v}_j^{(0)}, \mathbf{v}_j^{(1)}, ..., \mathbf{v}_j^{(m-1)}]$ contains m items as column vectors $\mathbf{v}_j^{(i)} \in \mathbb{R}^{k \times 1}$. All items in an observation are observed together and are considered to be jointly fashionable. Because T is a testing set, none of the observations it contains were used to train the model or tune its hyperparameters. Items of the same category i in all observations in

T are separated into m different inventories $T^{(i)} = \{\mathbf{v}_j^{(i)}\}_{j=0}^{n_v}$, which are sets of items and not sets of observations. Our ranking metric is defined relatively to the content and size of those m sets of items.

To formally define the RIRE metric, a distance metric must first be selected. Because of its simplicity and low-complexity, we use the cosine-similarity. Let $\mathbf{u} \in \mathbb{R}^k$ and $\mathbf{v} \in \mathbb{R}^k$ be two vectors of same dimensionality k. Their cosine-similarity is

$$c(\mathbf{u}, \mathbf{v}) = \frac{\mathbf{u}.\ \mathbf{v}}{||\mathbf{u}||\ .\ ||\mathbf{v}||}\ , \tag{5.1}$$

where c is a function $c : \mathbb{R}^k \times \mathbb{R}^k \to \mathbb{R}$ and its output is defined between -1 and 1. We then compute the similarity between the predicted ideal item with label i and each item of the temporary inventory $T^{(i)}$. Let $\hat{\mathbf{v}}_j^{(i)} \in \mathbb{R}^{k\times 1}$ be the ideal item predicted by a DAE, with corresponding observation $\mathbf{V}_j \in T$ and ground-truth item $\mathbf{v}_j^{(i)} \in T^{(i)}$. Note that we use the hat symbol on top of items to refer to predicted items $\hat{\mathbf{v}}_j^{(i)}$, just like we previously used the hat symbol on top of observations to refer to predicted observations $\hat{\mathbf{x}}_j$. Using the previously defined cosine-similarity metric, we can build a set of similarity score that are computed between the prediction $\hat{\mathbf{v}}_j^{(i)}$ and each item in $T^{(i)}$; that is

$$T_{c(j)}^{(i)} = \{s_l \mid s_l = c(\ \hat{\mathbf{v}}_j^{(i)}, \mathbf{v}_l^{(i)}\ )\ \}_{l=0}^{n_v}\ , \tag{5.2}$$

where subscript $c(j)$ denotes that the cosine similarity is taken with respect to the predicted ideal item $\hat{\mathbf{v}}_j^{(i)}$ with inventory index j.

We then order the set of similarity score $T_{c(j)}^{(i)}$ in decreasing order. Formally, a partial ordering on a set A according to a binary relation $\leq$ is a pair $(A, \leq)$ that satisfies reflexivity, antisymetry and transitivity. A partial ordering of $T_{c(j)}^{(i)}$ can therefore be defined as a pair $\Gamma_{c(j)}^{(i)} = (T_{c(j)}^{(i)},\ \leq)$. As previously stated, we are interested in the rank, or position, of the ground-truth item $\mathbf{v}_j^{(i)}$ in the partial ordering $\Gamma_{c(j)}^{(i)}$. To retrieve the rank of the ground-truth item $\mathbf{v}_j^{(i)}$ in $\Gamma_{c(j)}^{(i)}$, we use an index function $\pi_j(\ \Gamma_{c(j)}^{(i)}\ )$ that returns the index, or rank, of the ground item with inventory index j. Our final RIRE metric is obtained by normalizing the rank of item $\mathbf{v}_j^{(i)}$ in $\Gamma_{c(j)}^{(i)}$ by the size of inventory $T^{(i)}$:

$$RIRE(\ \hat{\mathbf{v}}_j^{(i)}, T^{(i)}\ ) = 1 - \frac{\pi_j(\ \Gamma_{c(j)}^{(i)}\ )}{|T^{(i)}| - 1} \tag{5.3}$$

Therefore, a RIRE score of 1 is produced when the ground-truth item $\mathbf{v}_j^{(i)}$ is ranked first in the partial ordering $\Gamma_{c(j)}^{(i)}$, and a RIRE score of 0 is produced when the ground-truth item $\mathbf{v}_j^{(i)}$ is ranked last. Note that the RIRE metric is simply used to assess the performance of our model and is not used as a loss function, contrary to the RMSE. This is because the RIRE metric, like all ranking metrics, is not differentiable. It is also useful to point out that computing the RIRE metric has a non-negligible computational cost. Indeed, Equation 5.3 is applied to each of the $m \times n_v$ items in the testing set T. Equation 5.3 includes n_v computation of cosine-similarity scores, which is in $\Theta(k)$, as well as a linear search for the rank of the ground-truth item which can be done in $\Theta(n_v)$. This gives an overall complexity of $\Theta(mn_v) \times \Theta(kn_v) \times \Theta(n_v) = \Theta(n_v^3 mk)$, which is cubic in the number of observations.

If we explained how the quality of the recommendation is measured, so far we did not explained how recommended items are selected after the model's deployment in production. Recall that we initially deal with m inventories as sets of items $I^{(i)} = \{\mathbf{p}_0^{(i)}, \mathbf{p}_1^{(i)}, ..., \mathbf{p}_{n_i}^{(i)}\}$ in which we pick items to recommend. Inventories $I^{(i)}$ are different from the temporary inventories $T^{(i)}$ that were created from the testing set T: T and its inventories of items $T^{(i)}$ are used to measure our model's performance, while inventories $I^{(i)}$ are used to make actual recommendations to users. Let $\mathbf{Q} \in \mathbb{R}^{k \times m}$ be a user query as a matrix of items $\mathbf{Q} = [\mathbf{q}^{(0)}, \mathbf{q}^{(1)}, ..., \mathbf{q}^{(m-1)}]$, and $\hat{\mathbf{q}}^{(i)}$ be the predicted item to replace item $\mathbf{q}^{(i)}$ from the query. The item $\dot{\mathbf{p}}^{(i)}$ that is most similar to $\hat{\mathbf{q}}^{(i)}$ and that we end up recommending is selected by searching the item in $I^{(i)}$ whose cosine-similarity is highest:

$$\dot{\mathbf{p}}^{(i)} = \underset{\mathbf{p}^{(i)} \in I^{(i)}}{\arg\max} \left(\frac{\mathbf{p}^{(i)} . \hat{\mathbf{q}}^{(i)}}{||\mathbf{p}^{(i)}|| \ . \ ||\hat{\mathbf{q}}^{(i)}||} \right) \tag{5.4}$$

This amounts to a naive nearest-neighbor search. Its complexity is in $\Theta(kn_i + n_i)$, but better bounds can be obtained using more advanced algorithms, some of the fastest being based on an approximation of the neirest-neighbor search.

5.3 Implementation

We train our model on a dataset of fashion outfit images, where each image contains exactly m items labeled with one of the m different labels, and no two items in a given observation can have the same label. Note that an important assumption is made when we select a fashion dataset: we assume that there exists a notion of joint fashionability between the apparels contained in each fashion outfit. This is not always the case, as a dataset of outfits could be made of apparels that are assembled chaotically. We decide to use the Polyvore Outfit dataset that was collected by [22] in 2017, and was later used in [55] as well. This dataset contains $21,889$ outfits from Polyvore.com, where each outfit contains on average 6.5 items. We select this dataset for several reasons. First, each outfit is made of high-quality images of both the outfit and of each apparel it contains. This removes the need to extract regions of the image that contain each apparel as a pre-processing step. Second, each outfit was specifically selected because of the positive feedback it received from the Polyvore.com community. This reinforces our assumption that outfits are stylish and therefore that the apparels they contain are not combined chaotically. In Figure 5.1, a comparison of the major available fashion datasets is shown.

Our implementation is made of three Python scripts. The first one extracts items inside images using either bounding boxes or polygon annotations. This script was not used for this specific implementation but was used on other datasets. The second script allows us to extract visual features from images using a modified classification model that will be presented below. The third script trains a DAE using feature vectors. The three scripts make use of various classes and functions that are imported from a Python module that was developed on the side. This makes our scripts more compact and improves readability.

As previously stated, we manipulate feature vectors representing each image, and not the image themselves. To extract feature vectors, we used a modified Resnet-50 model [24], a CNN classification model that was pre-trained on the ImageNet dataset [15] and that is provided by the Pytorch framework. The modification consists of removing the

Name	Year	#image	#label	#bbox	#landmark	#mask	#pair
CCP [73]	2014	2K	59	2K	×	1K	×
WTBI [32]	2015	425K	11	39K	×	×	39K
DARN	2015	182K	20	7K	×	×	91K
DeepFashion [45]	2016	800K	50	×	120K	×	251K
Modanet [75]	2018	55K	13	×	×	119K	×
FashionAI	2018	357K	41	×	100K	×	×
DeepFashion2 [19]	2019	491K	13	801K	801K	801K	873K
Fashionpedia [30]	2019	49K	27	49K	×	49K	×

Figure 5.1: Dataset comparison. Partially taken from [19]

last layer of the network and by using the values of the penultimate layer as output. More specifically, in that case, because images are projected into a latent space, feature vectors are also latent vectors. Although this feature extractor was trained on a large variety of different objects in ImageNet, modified image classifiers are known to be very good at extracting the visual features of other objects as well. Each image of apparel is encoded into a vector of 512 dimensions, normalized in the 0 to 1 range, and stored along with its observation index j and label index i into a JSON file for later processing.

Figure 5.2: Example of an outfit from the Polyvore Outfit dataset.

Items in the Polyvore Outfit dataset are labeled with one of 380 different categories, most of them being fashion accessories or even home furniture. In our case, we only want to use three categories, namely $\{top, bottom, shoe\}$. We could have chosen only two, but as outlined in Section 1.3, being able to recommend complementary items from more than two different categories is one of our key differentiators compared to existing approaches. We merged the original categories $\{"Tops", "Blouses", "Cardigans", "Sweaters", "T-Shirts", "Jackets", "Vests"\}$ into the new category "*top*", the original categories $\{"Skirts", "MiniSkirts", "KneeLengthSkirt", "LongSkirts", "Jeans", "Pants", "Shorts"\}$ into the new category "*bottom*", and the original categories $\{"Shoes", "Boots", "Pumps", "Sandals", "Flats", "FlipFlops", "Sneakers", "Slippers"\}$ into the new category "*shoe*". Other original categories are ignored. By only keeping observations who had at least one

of each of the three new categories, we obtained a total of $3,670$ usable observations. We first shuffled them before assigning 70% to a training set, 20% to a validation set, and 10% to a testing set.

Recall that, because when $\delta_{max} = 1$ there is m different ways a given observation could be corrupted, we have $n_S \times m$ potential corrupted observations to train with. We further randomize the sequence in which corrupted observations are fed to the model by shuffling the order in which labels are corrupted. At each epoch and for each observation, we create a list of indices from 0 to $m - 1$, where each element of this list corresponds to the label of the observation to be corrupted, and we randomly shuffle each of those lists. During training, we loop over the first element of each observation's list, then the second element, up to the m's elements of each observation's list. This ensures that all corrupted masks will be applied to each observation exactly once but in random order.

Because all variables are vectors of continuous values, we used the RMSE error defined in Equation 4.13 as our loss function. More specifically, we used the full RMSE, which is the RMSE computed between whole uncorrupted observations and the whole predicted observation. We experimented with a various number of layers but found that adding more layers only resulted in longer training time. As in Chapter 4, 4-layers (2 for the encoder, 2 for the decoder) seemed to be optimal. Both the training rate and the regularization weight were set to 0.0001. Again, the weights of the neural network were initialized using the He uniform method, and its biases were all set to zero. For monitoring purposes, we kept track of the full error, the partial error, and the RIRE score. We trained the DAE for 25 epochs using batches of 128 observations.

5.4 Results

As in Chapter 4, we measured both the partial error and the full error of the DAE. In each case, we further differentiated between the training error and the testing error.

(a) Full RMSE *(b)* Partial RMSE

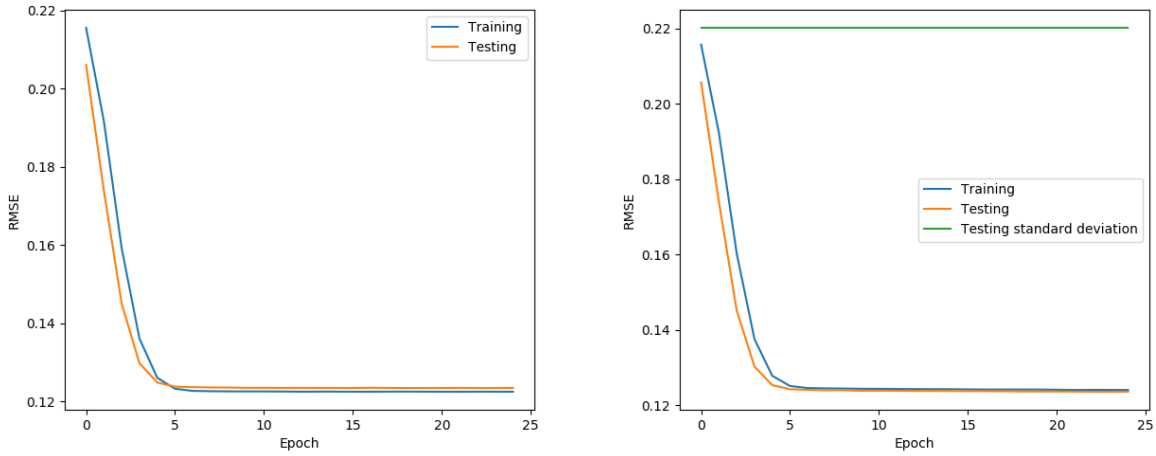

Figure 5.3: Full and Partial RMSE on Polyvore Outfit embeddings.

Figure 5.3 displays the partial and full RMSE on the training and testing set. We can see that both the partial and full RMSE quickly converge toward a minimum with

few epochs. We believe this speed of convergence to be due to different reasons than in Chapter 4. Because our dataset is made of images, extracting the abstract meaning that lies behind them requires an extensive use of non-linear transformations and convolutional layers. However, because our vectors of features are also latent vectors extracted from those images, most of this complexity was previously taken on by the feature extractor. One of the benefits of using latent vectors to represent images is that it represents them inside of a latent space. This latent space is such that images whose abstract content is similar are close to each other, whereas that is not necessarily the case in their original space. This is why it is not necessary to use very deep neural networks and a lot of training epochs to predict or reconstruct latent vectors using other latent vectors as input.

A first, simple way to verify that a model learns something is to compare its partial RMSE on the testing set to the standard deviation of the data. Here is why. Consider that the simplest approach to any regression task is to learn and predict the mean of the training data. This is what we refer to as the mean baseline. In that case, the RMSE of such a simple model will be equal to the standard deviation of the dataset. This is because the standard deviation of a random variable is identical to the RMSE between every possible value of this random variable and the expected value of this same random variable. Because we can assume that the training set has approximately the same empirical distribution as the testing set (due to both shuffling of the dataset before splitting and the large number of observations), so is their standard deviations. Therefore, any model whose RMSE is lower than the standard deviation of the data would perform better than the mean baseline. In Figure 5.3, we can see that the RMSE of our model goes below the standard deviation of the testing set, which is positive.

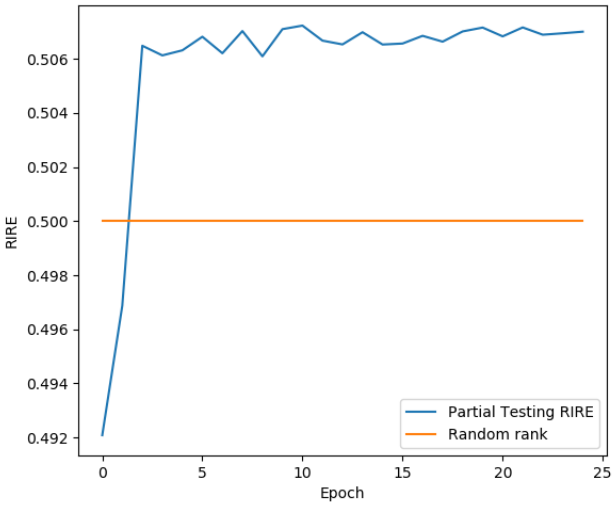

Figure 5.4: Partial RIRE score on the testing set of embeddings. Higher is better.

We now consider the quality of the recommendation relative to the size and content of our inventories of items. Figure 5.4 displays the mean RIRE score obtained on the testing set. A simple baseline for this metric is one that consists of randomly selecting an item from the testing set as a prediction of the ground-truth item. We denote this baseline as the random baseline. This would give, on average, a RIRE score of 0.5. Therefore, any model whose RIRE score on the testing set is above 0.5 can be considered as being superior when it comes to recommending jointly fashionable items. In Figure 5.4, we can

see that the mean RIRE score ends up being slightly above 0.5 at a value of 0.516. The DAE does not significantly outperform the random baseline. Therefore, we can hardly consider this as a sign of our model's capacity to recommend jointly fashionable items.

We provide an example of the predictions provided by the DAE. In Figure 5.5, we displayed 3 different predictions, each for one of the $m = 3$ different categories that could be recommended. The left column contains the incomplete query, the middle one the item that originally appears alongside the incomplete query, and the right column contains the item that was predicted by the DAE. In all cases, the ground truth item was not retrieved. However, one could judge that the predicted items are relatively compatible with their corresponding incomplete queries. For the first row, the predicted item is of the same style as its ground truth but its color is slightly different. In the second row, the style of the predicted item is clearly different than that of its ground truth item, however one could judge that the predicted item is not completely incompatible with its corresponding incomplete query. Finally, in the third row, the predicted item is black and not white like its ground truth item, however it is relatively compatible with its corresponding incomplete query. All in all, it is difficult to make any claim about the quality of the recommendation based on that kind of qualitative assessment.

Several reasons could be behind the poor performance of this model. Errors in our software implementation are also not ruled out. For instance, we believe this could be due to the necessity to conduct a nearest-neighbor search in high-dimensional space (we are searching through a list of $3,670$ 512-dimensional vectors). Indeed, one of the consequences of the curse of dimensionality is that two random high-dimensional vectors will tend to be orthogonal with high probability. This makes distance and similarity metrics meaningless as the dimensionality increases. This, on top of the noisy nature of the predicted items, could explain why the nearest-neighbor search fails to retrieve the ground-truth item. A solution could be to use latent vectors of lower dimensionality. A convolutional autoencoder could be trained from scratch with this characteristic in mind. A simpler, more traditional approach would be to apply Principal Component Analysis (PCA) to latent vectors as an additional pre-processing step to reduce their dimensionality. PCA is an algorithmic method that allows one to reduce the dimensionality of the data by maximizing the variance alongside each of the new dimensions, which are also known as principal components. This is done by applying a linear transformation to the original data. Principal components, i.e. new dimensions alongside which the variance is maximized, are computed and sorted according to the quantity of variance they contain. Generally, the number of principal components is selected to explain more than 95% of the total variance. As outline in Appendix 7.2.3, PCA has the benefit of forcing the new dimensions to be orthogonal from each other as opposed to linear autoencoders.

Another potential reason behind the poor performance that we obtained is the sample complexity of the task that we trying to solve, and which might be much higher than the 3670 observations that we have. Theoretically speaking, the sample complexity of a prediction algorithm is the minimum number of training samples necessary to obtain a prediction error that is greater than the best possible prediction error by a quantity of at most ϵ, with probability $1 - \gamma$. The notion of sample complexity, which is also known as the VC-dimension, is related to the well-known no-free-lunch theorem. It states that given an unbounded hypothesis class $\mathcal{H}$ (set of potential prediction functions to choose from, commonly being referred to as a model) supposed to solve any potential prediction task, the sample complexity is infinite. In other words, there is no such thing as a universal prediction algorithm that could solve all possible prediction tasks optimally. If exact

Figure 5.5: Example of an incomplete query with correponding ground truth item and predicted item

sample complexity can be found for simple toy models, only asymptotic bounds can be found for more complex models. Consider that a DAE is fundamentally a specific case of an MLP. It has been shown in [64] that an MLP with Heaviside activation functions has a sample complexity in $O((|\phi| + |\psi|) \log h)$, where $|\phi|$ the number of parameters in ϕ and h is the number of hidden neurons. However, activation functions are typically partially-differentiable functions like the sigmoid function. The sample complexity of general-purpose MLPs was found to be $O((|\phi| + |\psi|)^2 h^2)$. This bound is considered by practitioners as not tight as in practice very satisfying results can be obtained with a few thousand training samples in image classification tasks for example. In addition to considering both the complexity of the model and the complexity of the optimal prediction function, practitioners also reason by analogy with previous works. By considering the number of training samples that others used on similar prediction tasks, the sample complexity of a new task can be estimated.

To conclude, although our model was able to both reconstruct its input and predict missing variables better than the mean baseline, it failed to significantly outperform a random recommendation of complementary items. However, a quick qualitative assessment of the predictions does provide relatively satisfying results. We believe that further investigations are necessary.

5.5 Comparison

The DAE we used in this section is processing latent vectors extracted using a modified CNN classification model like in [68] [55] [22]. This is opposed to some previous methods [44] [26] [27] who used low-level visual features like SIFT or HOG, as well as other discrete attributes like color, compatible seasons, or apparel categories. Since the advent of convolution neural networks, it seems that extracting latent representation of fashion items using them is now considered to be the best approach. Note that this can either be done either separately as a pre-processing step before training the model, which is what we did, or directly inside of the model as in [41]. According to [41], learning latent representations of items jointly with the inference part of the model that recommends jointly fashionable items proved to be superior in terms of recommendation accuracy.

As demonstrated in Section 4, our method is versatile, as it can recommend any potentially missing item using a single model. Previously proposed models that we know of which could have this capacity given minor changes are [41] and [68]. Because the models proposed by [22] and [22] are LSTMs, they generate sequences of complementary items. Each new prediction is conditioned on the previously predicted items and the user query, which could potentially lead the model to "deviate" toward an arbitrary outfit. In comparison, our model can predict a whole set of jointly fashionable items using a single prediction, and recommended items will be exclusively conditioned on the items that are provided by the user.

Another difference between our approach and others is the robustness of our model, which allows us to recommend multiple items at the same time given various amounts of information at its disposal. Indeed, when predicting fashion items, a user might only have and provide 3 different items while the DAE can recommend any of $m \geq 3$ different categories of items. In our case, we trained our model with every potential corruption mask applied to all observations for a fixed maximum number of corrupted variables $\delta_{max} \leq m$. However, it remains to be seen how different training strategies exactly influence the capacity of DAEs to make robust predictions. For instance, is it really necessary to train with every potential corruption mask? Or is a subset of those potential corruption masks sufficient? And if yes, by how much can we reduce the size of this subset without impacting the prediction performance of the model? Those are unanswered questions.

Finally, compared to some of the most recent models that have been proposed to solve the problem of recommending jointly fashionable items, we believe our model to be one of the simplest. In the DAE itself, there is no need for very deep architectures, convolutional layers, or complex loss function. A pre-trained classification model can be used as a feature extractor. All items are represented using latent vectors which are easy to store and manipulate. Furthermore, DAEs are simple to implement, and the model itself does not take a long time to train.

5.6 Limitations

Apart from the poor performance that we obtained and discussed in the previous sections, other limitations must be taken into account. This is especially true when one is willing to use the proposed method in a real-life setting. Considering that a user would send an image of himself wearing different items, this would imply two different pre-processing steps. The first is the extraction of regions in the image that contains each item of

interest. The second is the extraction of visual features from the extracted items. Those two steps are not trivial and are essential to the prediction accuracy of the DAE.

In Chapter 5, both images of the whole outfit and separate images of each apparel contained in the outfit were provided. In practice, the region containing each item would first have to be predicted. In [29] and [74], bounding boxes were predicted by a pre-trained model. However, bounding boxes around items can potentially contain other items. Furthermore, some pixels inside bounding boxes belong to the background and are undesirable. This means that the feature extractor would extract visual features that do not pertain to the items that we are interested in. This can be alleviated using a segmentation model. A segmentation model, taking an image as input, will both detect items inside images and predict the class of each pixel in the image. It is then possible to only extract the pixels that belong to the item of interest. As an experiment, we fine-tuned Mask-RCNN [23], a semantic segmentation model from Facebook research, on the Modanet fashion dataset [75]. The objective of a semantic segmentation model is to classify each pixel using a limited set of labels, whereas instance segmentation models are semantic segmentation models who further differentiate between multiple instances of items from the same class. We obtained very satisfying results, even without tuning the hyperparameters.

Segmentation models are often evaluated using the mIoU (mean Intersection-Over-Union) which is defined as the ratio of the area of overlap by the area of union between the ground-truth area and the predicted area. The first proposed semantic segmentation model, the Fully Convolutional Network (FCN), was developed in 2015 by J.Long et al. [46]. FCN achieved a mIoU of 62.2% on the Pascal 2012 dataset. Numerous other architectures followed, including Mask-RCNN [23]. Mask-RCNN is an extension of Faster R-CNN, and relies on processing each detected Region-of-Interest (ROI) by using small Fully Connected Network (FCN) segmentation models for the mask prediction. According to the original paper, separating the prediction of the item mask from the prediction of the label of the mask resulted in a significant increase in prediction accuracy. Recently, Google released DeepLabV3+ [12] and managed to get a state of the art mioU of 89% on the Pascal 2012 dataset. We believe this prediction accuracy to be largely sufficient for a recommendation task, as both type-I and type-II errors are not dramatic when it comes to recommending fashion items. By using more training data and by eventually controlling the physical environment in which cameras are operating, we believe that those scores could be further increased.

A second limitation could come from the extraction of visual features that comes after the segmentation of items in images. Indeed, some feature extractors are better at extracting the most characterizing features of items than others. For practical reasons, modified pre-trained classification models are often used. This alleviates the need for training a new feature extractor from scratch, which is time-consuming. If this method provides very satisfying results in practice, it remains theoretically suboptimal. First, the pre-trained classification model was not exactly trained for extracting visual features. Second, the type of items it was trained to classify might differ from the ones we are interested in. A pre-trained model on ImageNet, for instance, would not be specialized but would rather be moderately good at extracting visual features from items of very different types. A better way would be to train from scratch a convolutional autoencoder like in [16], where the Resnet architecture defined in [24] was mixed with the autoencoder architecture defined in Appendix 7.2.3. Trained specifically on a dataset of fashion items, such a model should be theoretically better at extracting and organizing the most

characterizing features of fashion apparel.

Finally, the physical situation in which images are taken must not be underestimated. In a real-life scenario, people would be facing a camera that would take a picture of their outfit. As the saying goes: garbage in, garbage out. Ensuring the highest image quality taken by such cameras, and making sure that the scene is well lit and that colors are accurately captured would greatly help. On the hardware side of things, we now have access to high-definition cameras with definitions up to 4K (four times FullHD, which is 1920×1080 pixels per frame). It can generally be assumed that the person will be close to the camera and that the camera's optic, that is its field of view and focal length, can be engineered as needed.

To summarize, none of the listed limitations are unsurmountable. We believe the performance of the proposed method to be dependent on all of its constituents, and that improving each part would lead to improved results.

6. Conclusion

6.1 Possible extension to this work

Apart from finding the reason behind the poor performance that we obtained and trying to improve it, we see several possible extensions to this work.

The first would be to train a variational DAE (VDAE) [34] [28] instead of a simple DAE. Variational refers to the use of Variational Inference, a technique used to approximate intractable integrals that often appear in machine learning. A review of variational inference techniques can be found in [7]. The problem of vanilla DAEs is that they are only able to predict the mean of the estimated conditional probability distribution. In practice, however, there might be multiple good item candidates to complement a user query. Instead of just learning the set of parameters ψ of the encoding function f, a VDAE would also try to learn the parameters of a specific family of statistical distributions that would best fit values observed in the embedding layer. This distribution is often set to be a Gaussian. VDAEs [28], just like VAEs, are non-deterministic, and predictions are obtained by sampling their learned prior distribution. Therefore, one could obtain multiple good item candidates to complement a single user query.

The performance of VAEs, and therefore the performance of VDAEs as well, is limited by the family of statistical distribution that is being used. Take for example the case of a Gaussian distribution $\mathcal{N}(\mu, \sigma^2)$, which is the most widely used statistical distribution in VAEs. But what if the real probabilistic distribution of latent features was not Gaussian? This could have a dramatic impact on the prediction's performance of the VAE. Figure 4.4 shows how Gaussian distributions can sometimes fail to approximate the true probabilistic distribution of latent features. This, however, can be alleviated by the use of Generative Adversarial Networks (GAN). As its name suggests, GANs are generative models, meaning that they can create new data samples who do not belong to the set of training observations, but whose statistical distribution resembles that of the training set. Contrary to VAEs, GANs are not constrained by a specific family of statistical distribution: given enough expressivity (i.e. number of layers), they can approximate asymptotically any of them.

The DAE we used to recommend jointly fashionable items presents the same limitations as the one presented in Section 4 when multiple items are predicted at the same time. This arises from the exponential increase in the number of ways to corrupt input observations as the number of possible corrupted variables δ_{max} increases. A stochastic approach to training and validating a DAE when $\delta_{max} > 1$ could be beneficial. A DAE could be trained and validated on only a random subset of the potential corruption masks instead of all of them. Potentially, a random sampling of the corruption masks would provide a lower prediction accuracy. We hypothesize that one could balance the robustness of the DAE with its training time by doing so.

Finally, one must consider that, as δ_{max} increases, it becomes increasingly difficult for the model to make accurate predictions. Figure 4.3 shows that corrupted observations are further away from the embedded manifold than uncorrupted ones. In fact, this distance is not the same for all corrupted samples and increases along with the amount of corruption

that is applied. It is harder for a DAE to make accurate predictions when δ_{max} is high, which translates into a higher prediction error, as displayed in Figure 4.8. This begs the question; could we help the model learn faster? Whenever we train ourselves to become better at something, we never start with the hardest task. Instead, we start with an easy task and then tackle increasingly difficult ones as we improve. Similarly, we believe that first training with a small number of corrupted variables δ_{max}, resulting in a small average distance of corrupted observations from the embedded manifold, and then slowly increasing the amount of corruption as the prediction error gets lower could help a DAE learn faster.

6.2 Final words

We investigated the capability of DAEs to predict one or more of their missing input variables on a classical machine learning dataset. We showed that the sometimes limited prediction accuracy of our model was a consequence of it being a multi-task learner. We then tried to recommend jointly fashionable items using a DAE. The theoretical foundations of our method were proved. A qualitative assessment of the recommended items provided relatively satisfying results. Although both the partial and full errors of the DAE converged toward a minimum, indicating that the model was effectively learning, it failed to recommend the ground truth item significantly better than the random baseline when using our custom ranking metric. Potential reasons behind those results were provided. We remain convinced of the potential of the proposed method and believe that most of DAEs' potential remains untapped. Overall, additional work is necessary to investigate and improve the performance of the proposed model.

7. Appendix

7.1 Linear regression and the perceptron

Linear regression is the simplest kind of supervised regression problem. It assumes a linear relationship between predictor and target variables. Let $\mathbf{x} \in \mathbb{R}^k$ be an input vector of k predictor variables, and $y \in \mathbb{R}$ be a scalar outcome. We wish to model the relationship between the predictors and the outcome using a linear function $y = \mathbf{w}\mathbf{x} + b + \epsilon$, where $\mathbf{w} \in \mathbb{R}^k$ is a vector of weights, $b \in \mathbb{R}$ is a bias term, and ϵ is the error term which accounts for what cannot be explained by the linear model. The parameters $\mathbf{w}$ and b are selected as to minimize the error term ϵ.

Different estimation techniques can be used to estimate the parameters $\mathbf{w}$ and b, the most common one being Ordinary Least Square (OLS). OLS consists of minimizing the sum of squared errors. First observe that the bias term b can be merged into $\mathbf{w}$ by having $\mathbf{w} \in \mathbb{R}^{k+1}$ such that $w_0 = b$ and $\mathbf{x} \in \mathbb{R}^{k+1}$ with $x_0 = 1$. Now consider n_S different observations of predictors as a matrix $\mathbf{X} \in \mathbb{R}^{n_S \times (k+1)}$ and their corresponding outcomes as a vector $\mathbf{y} \in \mathbb{R}^{n_S}$. We must learn a matrix of weights $\mathbf{w} \in \mathbb{R}^{(k+1)}$ that is selected as to minimize the sum of squared errors. That is:

$$\mathbf{w} = \underset{\mathbf{w}}{\operatorname{argmin}} \, ||\mathbf{X}\mathbf{w} - \mathbf{y}||_2 \tag{7.1}$$

Because the loss function is convex, we can set its derivative with respect to $\mathbf{w}$ to zero and solve for $\mathbf{w}$. This yields the well-known closed-form solution

$$\mathbf{w} = (\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbf{y} \tag{7.2}$$

Depending on the number of predictors k and observations n_S, the closed-form solution can be computationally prohibitive. For instance, the $\mathbf{X}^T\mathbf{X}$ term has a storage complexity of $\Theta(n^2)$. Numerical stability issues can also arise. This is why iterative algorithms, such as Stochastic Gradient Descent (SGD) are sometimes used instead.

Now that we have introduced linear regression, we look into its classification counterpart, the perceptron. First introduced in 1958 by Frank Rosenblatt, an American psychologist, the perceptron is a binary classifier that assigns a given observation to one of two possible outcomes. It is obtained by applying an Heaviside step function $h : \mathbb{R} \to \{0, 1\}$ on top of the linear regression equation that we previously defined to obtain a binary vector. The Heaviside step function is defined as:

$$h(\mathbf{x}\mathbf{w}) = \begin{cases} 1 & \text{if } \mathbf{x}\mathbf{w} > 0 \\ 0 & \text{otherwise} \end{cases} \tag{7.3}$$

The bias term w_0 corresponds to the Heaviside boundary of decision between the positive and negative class, whereas other values of $\mathbf{w}$ weight the participation of predictors to the final result. It must be noted that n_S positive and negative observations

cannot always be linearly separable using a k-dimensional hyperplane. In that case, a probabilistic approach must be taken. If positive and negative observations are linearly separable however, then there might be many different valid solutions. Amongst those, only one maximizes the probability of correctly classifying unseen observations drawn from the same data generating distribution as the one from which training observations were drawn. The algorithm that yields such a solution is known as a Support Vector Machine (SVM) and works by maximizing the distance between the nearest data points and the boundary of decision, which is a hyperplane in the case of the perceptron. The resulting perceptron is also known as a perceptron of maximum stability.

We consider as an example the training of a perceptron on a training set of observations $S = \{(\mathbf{x}_j, t_j)\}_{j=0}^{n_S-1}$, where $\mathbf{x}_j \in \mathbb{R}^{k+1}$ and $t_j \in \{0, 1\}$ is the ground truth class for observation j. To train a perceptron on a linearly separable dataset, we first feed an observation to the perceptron and get its output y_j:

$$y_j = h(\mathbf{x}_j \mathbf{w}) \tag{7.4}$$

Let η be the training rate. Each feature i of the weight parameter $\mathbf{w}$ is updated as follow:

$$w_i \leftarrow w_i + \eta(t_j - y_j)x_{j,i} \ , \ \ \forall i, \ \ 0 \leq i \leq k+1 \tag{7.5}$$

We loop over all observations in S and apply equations 7.4 and 7.5 sequentially. Several epochs, that is repetitions of equation 7.4 and 7.5 over the whole set of observations S, might be needed depending on η. This training procedure is guaranteed to converged if observations in S are linearly separable. If they are not, then another approach must be used.

Linear regression and perceptrons are very simple in their ability to model the relationship between predictor and outcome variables. However, as we will later see, they form the fundamental building blocks of any neural network.

7.2 Neural Network

A neural network is a type of machine learning model that has the theoretical ability to approximate any mathematical function to an arbitrary degree of precision. Known since the 70's, it is only during the last decade, through multiple trials and errors and thanks to the drastic increase in computational capabilities, that their potential was revealed. There exists a wide variety of neural networks, also known as architectures, which can eventually be mixed as needed to benefit from their respective capabilities. In this section, we present the simplest neural network, the multi-layer perceptron (MLP), as well as the autoencoder architecture.

7.2.1 Multilayer Perceptron

The multi-layer perceptron (MLP) is the simplest kind of neural network. It is made of stacked layers of perceptrons whose Heaviside step functions are replaced by a partially differentiable function, which is also known as an activation function. There exist quantities of activation functions, each having their respective strengths and weaknesses. One

of the most popular one is the sigmoid, a specific case of the logistic function:

$$\sigma(x) = \frac{e^x}{1 + e^x} \tag{7.6}$$

The goal of an activation function is to make the model non-linear. This non-linearity allows the neural network to approximate any function, and therefore enables it to learn more complex relationships between the predictor and target variables. It should be noted that an MLP without activation functions is meaningless, as its sequence of linear layers would simplified into a single linear layer. Let the l subscript denotes the layer to which the specific parameter belongs, and L be the total number of layers. For simplicity, we consider that all layer l have the same dimensionality k. The neural network have an input $\mathbf{x} \in \mathbb{R}^k$ and an output $\mathbf{y} \in \mathbb{R}^k$. Each layer can be expressed in a matrix form where $\mathbf{W}_l \in \mathbb{R}^{k \times k}$ is the weight matrix, $\mathbf{b}_l \in \mathbb{R}^k$ is the bias of layer l, and $\mathbf{z}_l \in \mathbb{R}^k$ is the output of layer l. A neural network can be expressed as:

$$\begin{aligned} &\mathbf{z}_{l+1} = \sigma(\ \mathbf{W}_{l+1}\ \mathbf{z}_l + \mathbf{b}_{l+1}\ ) \\ &\text{where}\ \ \mathbf{z}_L = \mathbf{y} \\ &\text{and}\ \ \ \ \mathbf{z}_0 = \mathbf{x} \end{aligned} \tag{7.7}$$

We also define a loss function $\mathcal{L}$ to compute the prediction error of the neural network. On supervised learning tasks, the loss function is generally a discrepancy measure between the output of the neural network and what it should be, such that $\mathcal{L} : \mathbb{R}^k \times \mathbb{R}^k \to \mathbb{R}$. In the next section, we explain how this loss function can be used to learn the parameters $\mathbf{W}_l \in \mathbb{R}^k$ and $\mathbf{b}_l \in \mathbb{R}^k$ of a neural network. For a loss function to be used for this purpose, it must satisfy two requirements. First, it should be differentiable and be a function of the parameters. Second, we should be able to write it as a linear combination of each observation's error. Else, the SGD assumption that the added gradients of each observation's error would approximate the gradient of the combined errors wouldn't hold, and SGD wouldn't work.

Due to their non-linearity and their depth, training neural networks is not a trivial task. As we will now see, a specific technique is required in order for the training process to be computationally as efficient as possible.

7.2.2 How to train a Neural Network

This section provides a quick overview of how a simple MLP neural network can be trained efficiently using the backpropagation algorithm. The theoretical explanation that we provide is inspired by the reference textbook [58].

Before training a neural network, we first need to initialize its parameters. Correctly initializing the parameters can yield a much faster training time. It also prevents explosive and vanishing gradients, which would stop the training process. Weight matrices $\mathbf{W}_l$ of all layers cannot be set to zero because this would prevent neurons from a given layer to learn different things. Indeed, if several neurons participate equally to the output of their layers, then the gradient of the loss with respect to each of those neurons will be the same, and their values over time would remain the same as well.

Two characteristics of a good initialization method are sought out. This initialization method should be a function of the activation function that is used. First, the mean output of any layer should be zero if the activation function is symmetric about the

origin (like with the sigmoid) and equal to the mean of the input data otherwise (like with a ReLu activation function for instance). Second, the variance of each layer's output should be roughly the same for all layers. For instance, when a ReLu activation function is used, the He initialization method was proved to be optimal [25]. It consists of initializing the weights by drawing samples from a uniform distribution:

$$\mathbf{W}_l \sim \mathcal{U}(-\sqrt{12/(n_{in}+n_{out})}, \sqrt{12/(n_{in}+n_{out})}) \ , \tag{7.8}$$

where n_{in} and n_{out} respectively denote the input and output size of layer l.

Although neural networks are known since the 50's, training them has historically been a challenge due to a lack of computational capabilities and of an effective way to learn their parameters. It's only in 1975 that Paul J. Werbos first described in his Ph.D. thesis a method to efficiently train a neural network using the backpropagation algorithm. It was later formally described in [63]. The goal of training is to make the model "move" toward an ideally global, but sometimes just local, minimum of the loss function $\mathcal{L}$. One could update each parameter of the model by computing the gradient of the loss $\mathcal{L}$ with respect to each element of $\mathbf{W}_l$ and $\mathbf{b}_l$ and for each layer of the network using the chain rule $(f \circ g)' = (f' \circ g) \ . \ g'$. However, this approach would be inefficient as some computations would be repeated several times. The backpropagation algorithm allows us to efficiently update the parameters of each layer by using the recursive structure of the layers' respective errors.

We consider the backpropagation of a single observation as an input vector $\mathbf{x} \in \mathbb{R}^k$. Let $\mathcal{L}$ be some loss function of the parameters of the model, and whose exact nature is outside the scope of this section. As in Appendix 7.2.1, for simplicity we merge the bias term $\mathbf{b}_l$ of each layer l into its corresponding weight $\mathbf{W}_l$. We define the intermediate notation $\mathbf{y}_l = \mathbf{W}\mathbf{z}_{l-1}$, that is the output of layer l before the activation function, such that $\mathbf{z}_l = \sigma(\mathbf{y}_l)$. We first apply a forward pass to the input $\mathbf{x}$ through the network, and keep track of all outputs $\mathbf{z}_l$ as well as the derivative of the activation function σ' evaluated at $\mathbf{z}_l$, and that for each layer l. We can apply the chain rule to get the error of each layer up to the input $\mathbf{x}$:

$$\frac{\partial \mathcal{L}}{\partial \mathbf{x}} = \frac{\partial \mathcal{L}}{\partial \mathbf{z}_L} \cdot \frac{\partial \mathbf{z}_L}{\partial \mathbf{y}_L} \cdot \frac{\partial \mathbf{y}_L}{\partial \mathbf{z}_{L-1}} \cdot \frac{\partial \mathbf{z}_{L-1}}{\partial \mathbf{y}_{L-1}} \ ... \ \frac{\partial \mathbf{z}_1}{\partial \mathbf{y}_1} \cdot \frac{\partial \mathbf{y}_1}{\partial \mathbf{x}} \tag{7.9}$$

In this equation, we can replace what we already know. First we have the derivatives of the activation functions $\partial \mathbf{z}_l / \partial \mathbf{y}_l = \sigma'_l$ that we cached during the forward pass. Second, we have the rate of change between the output of layer l before the activation function and the output of the previous layer, which is simply $\partial \mathbf{y}_l / \partial \mathbf{z}_{l-1} = \mathbf{W}_l$.

$$\frac{\partial \mathcal{L}}{\partial \mathbf{x}} = \frac{\partial \mathcal{L}}{\partial \mathbf{z}_L} \ . \ \sigma'_L \ . \ \mathbf{W}_L \ . \ \sigma'_{L-1} \ . \ \mathbf{W}_{L-1} \ ... \ \sigma'_1 \ . \ \mathbf{W}_1 \tag{7.10}$$

Observe that the gradient of the loss $\mathcal{L}$ with respect to the input $\mathbf{x}$ is equal to the transpose of the derivative of the loss $\mathcal{L}$ with respect to the input $\mathbf{x}$:

$$\nabla_{\mathbf{x}} \mathcal{L} = (\frac{\partial \mathcal{L}}{\partial \mathbf{x}})^T = \mathbf{W}_1^T \ . \ \sigma'_1 \ ... \ \mathbf{W}_{L-1}^T \ . \ \sigma'_{L-1} \ . \ \mathbf{W}_L^T \ . \ \sigma'_L \ . \ \nabla_{\mathbf{z}_L} \mathcal{L} \tag{7.11}$$

$\nabla_{\mathbf{x}} \mathcal{L}$ can be seen as the contribution of the input $\mathbf{x}$ to the loss $\mathcal{L}$. Similarly, we are interested by the contribution of each layer l to the loss $\mathcal{L}$, which we loosely refer to as

the error of layer l. Consider the intermediate notation δ_l as the contribution of layer l to the loss $\mathcal{L}$. We can rewrite equation 7.11 as

$$\delta_l = \sigma'_l \,.\, \mathbf{W}_{l+1}^T \,...\, \mathbf{W}_{L-1}^T \,.\, \sigma'_{L-1} \,.\, \mathbf{W}_L^T \,.\, \sigma'_L \,.\, \nabla_{\mathbf{z}_L}\mathcal{L} \tag{7.12}$$

One can observe that computing δ_l necessarily implies to compute δ_{l+1}. We obtain a recursive definition of each layer's error δ_l:

$$\delta_l = \sigma'_l \,.\, \mathbf{W}_{l+1}^T \,.\, \delta_{l+1} \tag{7.13}$$

Therefore, we can compute the terms δ_l for all layers in a backward way, and use each δ_{l+1} to compute the next δ_l faster. This is the essence of the backpropagation algorithm. As we are interested in computing by how much each layer's weight $\mathbf{W}_l$ should change to minimize the loss function considered at $\mathbf{x}$, we compute the gradient of each layer's weight $\nabla_{\mathbf{W}_l}\mathcal{L}$ using

$$\nabla_{\mathbf{W}_l}\mathcal{L} = \delta_l \,.\, (\mathbf{z}_{l-1})^T \tag{7.14}$$

where the term $(\mathbf{z}_{l-1})^T$ removes the proportional effect that the activation $\mathbf{z}_{l-1}$ has on δ_l. Let η be the learning rate. Given that the gradient is oriented in the direction of steepest ascent, the weights are updated using SGD as to move toward a minimum of $\mathcal{L}$ as follow:

$$\mathbf{W}_l \leftarrow \mathbf{W}_l - \eta\nabla_{\mathbf{W}_l}\mathcal{L} \tag{7.15}$$

Now that we have explained how the weights are updated for each observation using backpropagation, we can summarize the training process of a neural network. It is an iterative process that consists of applying a forward pass to a batch of observations, computing its error, and backpropagating the error of the batch to each parameter of the neural network using equation 7.15. Repeating this process for all training observations is known as a training epoch. Depending on the curvature of the loss function and the training rate η, many epochs might be necessary before observing the convergence of the neural network toward a minimum of the loss function.

7.2.3 Autoencoder

Autoencoders are a specific architecture of neural networks whose objective is to reconstruct their input as accurately as possible on their output layer, and that are regularized as not to learn the identity function. The simplest type of regularization technique is to set $k' << k$, where k is the input layer size, and k' is the size of one of the hidden layer (generally assumed to be the middle one).

Autoencoders can be used for two different purposes. They can be used to reduce the dimensionality of their inputs by extracting the values of one of the middle hidden layers whose dimensionality k' is smaller than that of the input layer k. Extracted values are known as embeddings, or latent vectors, and are known to retain and organize most of the characterizing features of their inputs. Second, they can be used as a prediction model using one of their variant, the denoising autoencoder. Denoising autoencoders are used in this work and are defined in-depth in Section 4.1. In this section, we present a vanilla autoencoder that is regularized by a bottleneck hidden layer.

Consider a set of training sample $S = \{\mathbf{x}_j\}_{j=0}^{n_S-1}$, where each sample is a vector $\mathbf{x}_j \in \mathbb{R}^k$. Each observation is considered to be i.i.d distributed from an unknown distribution $\mathcal{D}$. An autoencoder is made of two functions, a encoder and a decoder. The encoder, as a

function $f_\phi : \mathbb{R}^k \to \mathbb{R}^{k'}$, parametrized by ϕ, produces a latent representation $\mathbf{y}_j \in \mathbb{R}^{k'}$, such that $f_\phi(\mathbf{x}_j) = \mathbf{y}_j$. The decoder, as a function $g_\psi : \mathbb{R}^{k'} \to \mathbb{R}^k$, is parametrized by ψ, and produce an output $\hat{\mathbf{x}}_j \in \mathbb{R}^k$, such that $g_\psi(\mathbf{y}_j) = \hat{\mathbf{x}}_j$.

Formally, f and g denote the architecture of the neural network, which includes the number of layers, their respective size, and how they are connected. However, it does not indicate which values those layers should have. This is why they need to be respectively parametrized by ϕ and ψ. ϕ and ψ are the sets of weights and biases used to initialize f and g. Formally, $\phi = \{(\mathbf{W}_l, \mathbf{b}_l)\}_{l=0}^{L/2}$, where L is the total number of layer and $\mathbf{W}_l$ is the weight parameter of layer l for function f, and conversely for function g with ψ. The parameters ϕ and ψ are updated as per the procedure defined in the previous section. To reduce the number of parameters that must be learned, $\phi = \psi^{-1}$ is often enforced (where ψ^{-1} denotes that the layers' order is reversed), which is known as having tied weights: this is because we can consider that $f = g^{-1}$.

Two problems might arise from not regularizing an autoencoder correctly. The first is the risk that the autoencoder might learn the identity function. The second, common to all machine learning models, is overfitting. Both are prevented by the use of regularization techniques. The bottleneck approach where $k' << k$ implies a lossy compression of the information. The smaller k', the more information will be lost about the input, and the harder it will be for the autoencoder to reconstruct the information. However, the latent representation $\mathbf{y}_j$ will be more compact and each of the features it contains will better characterize important aspects of its corresponding input $\mathbf{x}_j$. There exists many different regularization techniques. Essentially, what they all end up doing is forcing f and g to be as simple as possible while still allowing the model to generalize to unseen observations. Overall, the autoencoder framework can be summarized as

$$\begin{aligned} \mathbf{y}_j &= f(\mathbf{x}_j; \psi) \ , \\ \hat{\mathbf{x}}_j &= g(\mathbf{y}_j; \phi) \ , \\ \text{where} \quad \phi, \psi &\leftarrow \arg\min_{\phi,\psi} \frac{1}{n_S} \sum_{j=0}^{n_S - 1} \mathcal{L}(\mathbf{x}_j, \ g(f(\mathbf{x}_j; \psi); \phi)) \end{aligned} \tag{7.16}$$

where $\mathcal{L}$ is some arbitrary loss function. The loss function of a DAE is generally set to minimize some discrepancy measure between the input vector and the output vector.

There is an interesting relationship between autoencoders and Principal Component Analysis (PCA), another method of dimensionality reduction that is linear. It has been proved by Bourlard and Kamp in 1988 [8] that a 2-layer autoencoder without activation functions is equivalent to SVD, and therefore to PCA since SVD is the commonly used way to compute PCA. PCA's objective is to find the best linear subspace of lower dimensionality such that the variance alongside the dimensions of the new subspace is maximized. In theory, both PCA and linear autoencoders move toward an optimal subspace which maximizes the captured variance. But in practice, linear autoencoders might find several different bases for such optimal subspace, while PCA has the property to always end up with a unique and orthogonal basis, meaning that the features in this subspace are independent.

Bibliography

[1] Kush Agrawal. To study the phenomenon of the moravec's paradox, 2010.

[2] Guillaume Alain and Yoshua Bengio. What regularized auto-encoders learn from the data generating distribution. In *International Conference on Learning Representations (ICLR'2013)*, 2013.

[3] Connelly Barnes, Eli Shechtman, Adam Finkelstein, and Dan B Goldman. Patchmatch: A randomized correspondence algorithm for structural image editing. *ACM Trans. Graph.*, 28(3), July 2009.

[4] Justin Basilico and Thomas Hofmann. Unifying collaborative and content-based filtering. In *Proceedings of the Twenty-First International Conference on Machine Learning*, ICML '04, page 9, New York, NY, USA, 2004. Association for Computing Machinery.

[5] Herbert Bay, Andreas Ess, Tinne Tuytelaars, and Luc Van Gool. Speeded-up robust features (surf). *Computer vision and image understanding*, 110(3):346–359, 2008.

[6] BigsnarfDude. Denoising autoencoder pytorch cuda, 2017.

[7] David M. Blei, Alp Kucukelbir, and Jon D. McAuliffe. Variational inference: A review for statisticians. *Journal of the American Statistical Association*, 112(518):859–877, Feb 2017.

[8] Herve Bourlard and Y Kamp. Auto-association by multilayer perceptrons and singular value decomposition. *Biological cybernetics*, 59:291–4, 02 1988.

[9] John S. Breese, David Heckerman, and Carl Myers Kadie. Empirical analysis of predictive algorithms for collaborative filtering. *ArXiv*, abs/1301.7363, 1998.

[10] Fidel Cacheda, Víctor Carneiro, Diego Fernández, and Vreixo Formoso. Comparison of collaborative filtering algorithms: Limitations of current techniques and proposals for scalable, high-performance recommender systems. *ACM Trans. Web*, 5(1), February 2011.

[11] Rich Caruana. Multitask learning. *Machine Learning*, 28, 07 1997.

[12] Liang-Chieh Chen, Yukun Zhu, George Papandreou, Florian Schroff, and Hartwig Adam. Encoder-decoder with atrous separable convolution for semantic image segmentation, 2018.

[13] The Business Research Company. Clothing and apparel global market opportunities and strategies to 2022. Technical report, 2019.

[14] Maurizio Ferrari Dacrema, Paolo Cremonesi, and Dietmar Jannach. Are we really making much progress? a worrying analysis of recent neural recommendation approaches. *Proceedings of the 13th ACM Conference on Recommender Systems*, Sep 2019.

[15] J. Deng, W. Dong, R. Socher, L. Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *2009 IEEE Conference on Computer Vision and Pattern Recognition*, pages 248–255, 2009.

[16] Jianfeng Dong, Xiao-Jiao Mao, Chunhua Shen, and Yu-Bin Yang. Learning deep representations using convolutional auto-encoders with symmetric skip connections, 2016.

[17] Vincent Dumoulin, Ishmael Belghazi, Ben Poole, Olivier Mastropietro, Alex Lamb, Martin Arjovsky, and Aaron Courville. Adversarially learned inference, 2016.

[18] Vincent Dumoulin, Ishmael Belghazi, Ben Poole, Olivier Mastropietro, Alex Lamb, Martin Arjovsky, and Aaron Courville. Adversarially learned inference presentation, 2016.

[19] Yuying Ge, Ruimao Zhang, Lingyun Wu, Xiaogang Wang, Xiaoou Tang, and Ping Luo. Deepfashion2: A versatile benchmark for detection, pose estimation, segmentation and re-identification of clothing images, 2019.

[20] Rainer Gemulla, Peter Haas, Yannis Sismanis, Christina Teflioudi, and Faraz Makari. Large-scale matrix factorization with distributed stochastic gradient descent. 01 2011.

[21] Ian J. Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial networks, 2014.

[22] Xintong Han, Zuxuan Wu, Yu-Gang Jiang, and Larry S. Davis. Learning fashion compatibility with bidirectional lstms. *Proceedings of the 2017 ACM on Multimedia Conference - MM '17*, 2017.

[23] Kaiming He, Georgia Gkioxari, Piotr Dollár, and Ross B. Girshick. Mask R-CNN. *CoRR*, abs/1703.06870, 2017.

[24] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition, 2015.

[25] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification, 2015.

[26] Yang Hu, Xi Yi, and Larry S. Davis. Collaborative fashion recommendation: A functional tensor factorization approach. In *Proceedings of the 23rd ACM International Conference on Multimedia*, MM '15, page 129–138, New York, NY, USA, 2015. Association for Computing Machinery.

[27] Yang Hu, Xi Yi, and Larry S. Davis. Collaborative fashion recommendation: A functional tensor factorization approach. In *Proceedings of the 23rd ACM International Conference on Multimedia*, MM '15, pages 129–138, New York, NY, USA, 2015. ACM.

[28] Daniel Jiwoong Im, Sungjin Ahn, Roland Memisevic, and Yoshua Bengio. Denoising criterion for variational auto-encoding framework, 2015.

[29] Tomoharu Iwata, Shinji Watanabe, and Hiroshi Sawada. Fashion coordinates recommender system using photographs from fashion magazines. In *Proceedings of the Twenty-Second International Joint Conference on Artificial Intelligence - Volume Volume Three*, IJCAI'11, page 2262–2267. AAAI Press, 2011.

[30] Menglin Jia, Mengyun Shi, Mikhail Sirotenko, Yin Cui, Bharath Hariharan, Claire Cardie, and Serge Belongie. The fashionpedia ontology and fashion segmentation dataset.

[31] Rudolph Emil Kalman. A new approach to linear filtering and prediction problems. *Transactions of the ASME–Journal of Basic Engineering*, 82(Series D):35–45, 1960.

[32] M Hadi Kiapour, Xufeng Han, Svetlana Lazebnik, Alexander C Berg, and Tamara L Berg. Where to buy it: Matching street clothing photos in online shops. In *2015 IEEE International Conference on Computer Vision (ICCV)*, pages 3343–3351. IEEE, 2015.

[33] Juhwan Kim, Seokyong Song, and Son-Cheol Yu. Denoising auto-encoder based image enhancement for high resolution sonar image. *2017 IEEE Underwater Technology (UT)*, pages 1–5, 2017.

[34] Diederik P Kingma and Max Welling. Auto-encoding variational bayes, 2013.

[35] Yehuda Koren. Factorization meets the neighborhood: A multifaceted collaborative filtering model. In *Proceedings of the 14th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '08, page 426–434, New York, NY, USA, 2008. Association for Computing Machinery.

[36] Yehuda Koren. Collaborative filtering with temporal dynamics. In *Proceedings of the 15th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '09, page 447–456, New York, NY, USA, 2009. Association for Computing Machinery.

[37] Abhishek Kumar and Hal Daume III. Learning task grouping and overlap in multi-task learning, 2012.

[38] Christian Ledig, Lucas Theis, Ferenc Huszar, Jose Caballero, Andrew Cunningham, Alejandro Acosta, Andrew Aitken, Alykhan Tejani, Johannes Totz, Zehan Wang, and Wenzhe Shi. Photo-realistic single image super-resolution using a generative adversarial network, 2016.

[39] Jure Leskovec, Anand Rajaraman, and Jeffrey David Ullman. *Mining of Massive Datasets*. Cambridge University Press, USA, 2nd edition, 2014.

[40] Boduo Li, Sandeep Tata, and Yannis Sismanis. Sparkler: Supporting large-scale matrix factorization. pages 625–636, 03 2013.

[41] Yujie Lin, Pengjie Ren, Zhumin Chen, Zhaochun Ren, Jun Ma, and Maarten de Rijke. Improving outfit recommendation with co-supervision of fashion generation. In *The World Wide Web Conference*, WWW '19, pages 1095–1105, New York, NY, USA, 2019. ACM.

[42] G. Linden, B. Smith, and J. York. Amazon.com recommendations: item-to-item collaborative filtering. *IEEE Internet Computing*, 7(1):76–80, 2003.

[43] Guilin Liu, Fitsum A. Reda, Kevin J. Shih, Ting-Chun Wang, Andrew Tao, and Bryan Catanzaro. Image inpainting for irregular holes using partial convolutions, 2018.

[44] Si Liu, Jiashi Feng, Zheng Song, Tianzhu Zhang, Hanqing Lu, Changsheng Xu, and Shuicheng Yan. Hi, magic closet, tell me what to wear! In *Proceedings of the 20th ACM International Conference on Multimedia*, MM '12, pages 619–628, New York, NY, USA, 2012. ACM.

[45] Ziwei Liu, Ping Luo, Shi Qiu, Xiaogang Wang, and Xiaoou Tang. Deepfashion: Powering robust clothes recognition and retrieval with rich annotations. pages 1096–1104, 06 2016.

[46] Jonathan Long, Evan Shelhamer, and Trevor Darrell. Fully convolutional networks for semantic segmentation, 2014.

[47] Yongxi Lu, Abhishek Kumar, Shuangfei Zhai, Yu Cheng, Tara Javidi, and Rogerio Feris. Fully-adaptive feature sharing in multi-task networks with applications in person attribute classification, 2016.

[48] Ian MacKenzie, Chris Meyer, and Steve Noble. How retailers can keep up with consumers, 2013.

[49] Alireza Makhzani, Jonathon Shlens, Navdeep Jaitly, Ian Goodfellow, and Brendan Frey. Adversarial autoencoders, 2015.

[50] Xiao-Jiao Mao, Chunhua Shen, and Yu-Bin Yang. Image restoration using convolutional auto-encoders with symmetric skip connections, 2016.

[51] Jorge Martinez-Gil. Sift: An algorithm for extracting structural information from taxonomies, 2016.

[52] Llew Mason, Jonathan Baxter, Peter Bartlett, and Marcus Frean. Boosting algorithms as gradient descent. In *Proceedings of the 12th International Conference on Neural Information Processing Systems*, NIPS'99, page 512–518, Cambridge, MA, USA, 1999. MIT Press.

[53] McKinsey&Company. The state of fashion 2019. Technical report, 2019.

[54] Prem Melville, Raymond J Mooney, and Ramadass Nagarajan. Content-boosted collaborative filtering for improved recommendations. 2002.

[55] Takuma Nakamura and Ryosuke Goto. Outfit generation and style extraction via bidirectional lstm and autoencoder. *ArXiv*, abs/1807.03133, 2018.

[56] Warwick Nash, T.L. Sellers, S.R. Talbot, A.J. Cawthorn, and W.B. Ford. 7he population biology of abalone (haliotis species) in tasmania. i. blacklip abalone (h. rubra) from the north coast and islands of bass strait. *Sea Fisheries Division, Technical Report No*, 48, 01 1994.

[57] Tung Nguyen, Kazuki Mori, and Ruck Thawonmas. Image colorization using a deep convolutional neural network, 2016.

[58] Michael A. Nielsen. Neural networks and deep learning, 2018.

[59] Steffen Rendle, Christoph Freudenthaler, Zeno Gantner, and Lars Schmidt-Thieme. Bpr: Bayesian personalized ranking from implicit feedback, 2012.

[60] Paul Resnick, Neophytos Iacovou, Mitesh Suchak, Peter Bergstrom, and John Riedl. Grouplens: An open architecture for collaborative filtering of netnews. In *Proceedings of the 1994 ACM Conference on Computer Supported Cooperative Work*, CSCW '94, page 175–186, New York, NY, USA, 1994. Association for Computing Machinery.

[61] Stephen Robertson. Understanding inverse document frequency: On theoretical arguments for idf. *Journal of Documentation - J DOC*, 60:503–520, 10 2004.

[62] Henry A Rowley, Shumeet Baluja, and Takeo Kanade. Neural network-based face detection. *IEEE Transactions on pattern analysis and machine intelligence*, 20(1):23–38, 1998.

[63] David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. Learning representations by back-propagating errors. *nature*, 323(6088):533–536, 1986.

[64] Akito Sakurai. Tight bounds for the vc-dimension of piecewise polynomial networks. In *Advances in neural information processing systems*, pages 323–329, 1999.

[65] Suvash Sedhain, Aditya Krishna Menon, Scott Sanner, and Lexing Xie. Autorec: Autoencoders meet collaborative filtering. In *Proceedings of the 24th International Conference on World Wide Web*, WWW '15 Companion, page 111–112, New York, NY, USA, 2015. Association for Computing Machinery.

[66] Shai Shalev-Shwartz and Shai Ben-David. *Understanding Machine Learning: From Theory to Algorithms*. Cambridge University Press, USA, 2014.

[67] Friedrich Sommer and Thomas Wennekers. Models of distributed associative memory networks in the brain. *Theory in Biosciences*, 122:55–69, 05 2003.

[68] Xuemeng Song, Fuli Feng, Jinhuan Liu, Zekun Li, Liqiang Nie, and Jun Ma. Neurostylist: Neural compatibility modeling for clothing matching. In *Proceedings of the 25th ACM International Conference on Multimedia*, MM '17, pages 753–761, New York, NY, USA, 2017. ACM.

[69] Sho Sonoda and Noboru Murata. Transportation analysis of denoising autoencoders: a novel method for analyzing deep neural networks, 2017.

[70] P. Szendro, G. Vincze, and Andras Szasz. Bio-response to white noise excitation. *Electromagnetic Biology and Medicine*, 20:215–229, 06 2001.

[71] Pascal Vincent, Hugo Larochelle, Yoshua Bengio, and Pierre-Antoine Manzagol. Extracting and composing robust features with denoising autoencoders. In *Proceedings of the 25th International Conference on Machine Learning*, ICML '08, page 1096–1103, New York, NY, USA, 2008. Association for Computing Machinery.

[72] Kiri Wagstaff. Machine learning that matters, 2012.

[73] Wei Yang, Ping Luo, and Liang Lin. Clothing co-parsing by joint image segmentation and labeling. *2014 IEEE Conference on Computer Vision and Pattern Recognition*, Jun 2014.

[74] Yi Yang and Deva Ramanan. Articulated pose estimation with flexible mixtures-of-parts. In *CVPR 2011*, pages 1385–1392. IEEE, 2011.

[75] Shuai Zheng, Fan Yang, M. Hadi Kiapour, and Robinson Piramuthu. Modanet: A large-scale street fashion dataset with polygon annotations, 2018.

[76] Ziwei Zhu, Jianling Wang, and James Caverlee. Improving top-k recommendation via jointcollaborative autoencoders. In *The World Wide Web Conference*, WWW '19, page 3483–3482, New York, NY, USA, 2019. Association for Computing Machinery.